Tcl8.4/Tk8.4 Manual
# Table of Contents

**Tcl8.4/Tk8.4 Manual** ......................................................................................................................... 1

**Tcl/Tk Applications** ................................................................................................................................ 2

- **tclsh** ........................................................................................................................................... 3
  - NAME .......................................................................................................................................... 3
  - SYNOPSIS .............................................................................................................................. 3
  - DESCRIPTION ........................................................................................................................ 3
  - SCRIPT FILES ........................................................................................................................ 3
  - VARIABLES .......................................................................................................................... 4
  - PROMPTS ................................................................................................................................... 5
  - STANDARD CHANNELS ........................................................................................................ 5
  - SEE ALSO .................................................................................................................................. 5
  - KEYWORDS ............................................................................................................................ 5

- **wish** ............................................................................................................................................ 6
  - NAME ......................................................................................................................................... 6
  - SYNOPSIS ............................................................................................................................. 6
  - OPTIONS .................................................................................................................................. 6
  - DESCRIPTION ........................................................................................................................ 7
  - OPTIONS .................................................................................................................................. 7
  - APPLICATION NAME AND CLASS ....................................................................................... 8
  - VARIABLES ........................................................................................................................... 8
  - SCRIPT FILES ........................................................................................................................ 8
  - PROMPTS .................................................................................................................................. 9
  - KEYWORDS ............................................................................................................................. 9

**Tcl Commands** ..................................................................................................................................... 10

- **safe** ........................................................................................................................................... 11
  - NAME ......................................................................................................................................... 12
  - SYNOPSIS .................................................................................................................................. 12
  - OPTIONS .................................................................................................................................... 12
  - DESCRIPTION .......................................................................................................................... 12
  - COMMANDS ............................................................................................................................ 12
  - OPTIONS .................................................................................................................................... 14
  - ALIASES ..................................................................................................................................... 15
  - SECURITY .................................................................................................................................... 15
  - SEE ALSO .................................................................................................................................. 16
  - KEYWORDS ................................................................................................................................ 16

- **Tcl** ............................................................................................................................................... 18
  - NAME ......................................................................................................................................... 18
  - SYNOPSIS .................................................................................................................................. 18
  - DESCRIPTION .......................................................................................................................... 19

- **after** ........................................................................................................................................... 22
  - NAME ......................................................................................................................................... 22
# Table of Contents

**Tcl Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>append</td>
<td>25</td>
</tr>
<tr>
<td>array</td>
<td>26</td>
</tr>
<tr>
<td>bgerror</td>
<td>30</td>
</tr>
<tr>
<td>binary</td>
<td>32</td>
</tr>
<tr>
<td>break</td>
<td>44</td>
</tr>
</tbody>
</table>

---

**SEE ALSO**

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>append</td>
<td>25</td>
</tr>
<tr>
<td>array</td>
<td>26</td>
</tr>
<tr>
<td>bgerror</td>
<td>30</td>
</tr>
<tr>
<td>binary</td>
<td>32</td>
</tr>
<tr>
<td>break</td>
<td>44</td>
</tr>
</tbody>
</table>

---

**EXAMPLES**

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>append</td>
<td>25</td>
</tr>
<tr>
<td>array</td>
<td>26</td>
</tr>
<tr>
<td>bgerror</td>
<td>30</td>
</tr>
<tr>
<td>binary</td>
<td>32</td>
</tr>
<tr>
<td>break</td>
<td>44</td>
</tr>
</tbody>
</table>

---

**PLATFORM ISSUES**

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>append</td>
<td>25</td>
</tr>
<tr>
<td>array</td>
<td>26</td>
</tr>
<tr>
<td>bgerror</td>
<td>30</td>
</tr>
<tr>
<td>binary</td>
<td>32</td>
</tr>
<tr>
<td>break</td>
<td>44</td>
</tr>
</tbody>
</table>

---

**BINARY SCAN**

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>append</td>
<td>25</td>
</tr>
<tr>
<td>array</td>
<td>26</td>
</tr>
<tr>
<td>bgerror</td>
<td>30</td>
</tr>
<tr>
<td>binary</td>
<td>32</td>
</tr>
<tr>
<td>break</td>
<td>44</td>
</tr>
</tbody>
</table>

---

**BINARY FORMAT**

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>append</td>
<td>25</td>
</tr>
<tr>
<td>array</td>
<td>26</td>
</tr>
<tr>
<td>bgerror</td>
<td>30</td>
</tr>
<tr>
<td>binary</td>
<td>32</td>
</tr>
<tr>
<td>break</td>
<td>44</td>
</tr>
</tbody>
</table>

---

**DESCRIPTION**

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>append</td>
<td>25</td>
</tr>
<tr>
<td>array</td>
<td>26</td>
</tr>
<tr>
<td>bgerror</td>
<td>30</td>
</tr>
<tr>
<td>binary</td>
<td>32</td>
</tr>
<tr>
<td>break</td>
<td>44</td>
</tr>
</tbody>
</table>

---

**EXAMPLE**

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>append</td>
<td>25</td>
</tr>
<tr>
<td>array</td>
<td>26</td>
</tr>
<tr>
<td>bgerror</td>
<td>30</td>
</tr>
<tr>
<td>binary</td>
<td>32</td>
</tr>
<tr>
<td>break</td>
<td>44</td>
</tr>
</tbody>
</table>

---

**KEYWORDS**

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>append</td>
<td>25</td>
</tr>
<tr>
<td>array</td>
<td>26</td>
</tr>
<tr>
<td>bgerror</td>
<td>30</td>
</tr>
<tr>
<td>binary</td>
<td>32</td>
</tr>
<tr>
<td>break</td>
<td>44</td>
</tr>
</tbody>
</table>
# Table of Contents

**Tcl Commands**

- **KEYWORDS .......................................................................................................................... 44**
  - catch ........................................................................................................................................... 45
    - NAME ...................................................................................................................................... 45
    - SYNOPSIS ............................................................................................................................ 45
    - DESCRIPTION .......................................................................................................................... 45
    - EXAMPLES ............................................................................................................................. 45
    - SEE ALSO ............................................................................................................................... 46
    - KEYWORDS ............................................................................................................................ 46
  - cd ................................................................................................................................................ 47
    - NAME ...................................................................................................................................... 47
    - SYNOPSIS ............................................................................................................................ 47
    - DESCRIPTION .......................................................................................................................... 47
    - EXAMPLES ............................................................................................................................. 47
    - SEE ALSO ............................................................................................................................... 47
    - KEYWORDS ............................................................................................................................ 47
  - clock .......................................................................................................................................... 48
    - NAME ...................................................................................................................................... 49
    - SYNOPSIS ............................................................................................................................ 49
    - DESCRIPTION .......................................................................................................................... 49
    - SEE ALSO ............................................................................................................................... 53
    - KEYWORDS ............................................................................................................................ 53
  - close ......................................................................................................................................... 54
    - NAME ...................................................................................................................................... 54
    - SYNOPSIS ............................................................................................................................ 54
    - DESCRIPTION .......................................................................................................................... 54
    - EXAMPLE .................................................................................................................................. 54
    - SEE ALSO ............................................................................................................................... 55
    - KEYWORDS ............................................................................................................................ 55
  - concat ...................................................................................................................................... 56
    - NAME ...................................................................................................................................... 56
    - SYNOPSIS ............................................................................................................................ 56
    - DESCRIPTION .......................................................................................................................... 56
    - EXAMPLES ............................................................................................................................. 56
    - SEE ALSO ............................................................................................................................... 56
    - KEYWORDS ............................................................................................................................ 56
  - continue ................................................................................................................................... 58
    - NAME ...................................................................................................................................... 58
    - SYNOPSIS ............................................................................................................................ 58
    - DESCRIPTION .......................................................................................................................... 58
    - EXAMPLE .................................................................................................................................. 58
    - SEE ALSO ............................................................................................................................... 58
    - KEYWORDS ............................................................................................................................ 58
# Table of Contents

## Tcl Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dde</td>
<td>59</td>
</tr>
<tr>
<td>encoding</td>
<td>63</td>
</tr>
<tr>
<td>eof</td>
<td>65</td>
</tr>
<tr>
<td>error</td>
<td>67</td>
</tr>
<tr>
<td>eval</td>
<td>69</td>
</tr>
<tr>
<td>exec</td>
<td>70</td>
</tr>
</tbody>
</table>

**Tcl8.4/Tk8.4 Manual**

iv
# Table of Contents

**Tcl Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>fcopy</td>
<td>95</td>
</tr>
<tr>
<td>fconfigure</td>
<td>90</td>
</tr>
<tr>
<td>fblocked</td>
<td>88</td>
</tr>
<tr>
<td>exit</td>
<td>77</td>
</tr>
<tr>
<td>expr</td>
<td>79</td>
</tr>
<tr>
<td>fcopy</td>
<td>95</td>
</tr>
<tr>
<td>fconfigure</td>
<td>90</td>
</tr>
<tr>
<td>fblocked</td>
<td>88</td>
</tr>
<tr>
<td>exit</td>
<td>77</td>
</tr>
<tr>
<td>expr</td>
<td>79</td>
</tr>
</tbody>
</table>

---

**PORTABILITY ISSUES** ............................................................... 73
**UNIX EXAMPLES** ....................................................................... 75
**WINDOWS EXAMPLES** .................................................................. 76
**SEE ALSO** ............................................................................... 76
**KEYWORDS** ............................................................................... 76

---

**NAME** ..................................................................................... 77
**SYNOPSIS** ............................................................................... 77
**DESCRIPTION** ......................................................................... 77
**EXAMPLE** ............................................................................... 77
**SEE ALSO** ............................................................................... 77
**KEYWORDS** ............................................................................... 77

---

**NAME** ..................................................................................... 80
**SYNOPSIS** ............................................................................... 80
**DESCRIPTION** ......................................................................... 80
**OPERANDS** ............................................................................. 80
**OPERATORS** ............................................................................ 81
**MATH FUNCTIONS** ................................................................... 83
**TYPES, OVERFLOW, AND PRECISION** ...................................... 84
**STRING OPERATIONS** ............................................................... 85
**PERFORMANCE CONSIDERATIONS** ............................................. 85
**EXAMPLES** ............................................................................. 86
**SEE ALSO** ............................................................................... 87
**KEYWORDS** ............................................................................... 87

---

**NAME** ..................................................................................... 88
**SYNOPSIS** ............................................................................... 88
**DESCRIPTION** ......................................................................... 88
**EXAMPLE** ............................................................................... 88
**SEE ALSO** ............................................................................... 89
**KEYWORDS** ............................................................................... 89

---

**NAME** ..................................................................................... 90
**SYNOPSIS** ............................................................................... 90
**DESCRIPTION** ......................................................................... 90
**STANDARD CHANNELS** .......................................................... 93
**EXAMPLES** ............................................................................. 93
**SEE ALSO** ............................................................................... 94
**KEYWORDS** ............................................................................... 94

---

**NAME** ..................................................................................... 95
## Table of Contents

### Tcl Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td>95</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>95</td>
</tr>
<tr>
<td>EXAMPLE</td>
<td>96</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>97</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>97</td>
</tr>
<tr>
<td>filename</td>
<td>98</td>
</tr>
<tr>
<td>NAME</td>
<td>99</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>99</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>99</td>
</tr>
<tr>
<td>PORTABILITY ISSUES</td>
<td>104</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>104</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>105</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>105</td>
</tr>
<tr>
<td>file</td>
<td>106</td>
</tr>
<tr>
<td>NAME</td>
<td>106</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>106</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>106</td>
</tr>
<tr>
<td>EXAMPLE</td>
<td>107</td>
</tr>
<tr>
<td>CREDITS</td>
<td>107</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>107</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>107</td>
</tr>
<tr>
<td>filename</td>
<td>108</td>
</tr>
<tr>
<td>NAME</td>
<td>108</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>109</td>
</tr>
<tr>
<td>PATH TYPES</td>
<td>109</td>
</tr>
<tr>
<td>PATH SYNTAX</td>
<td>109</td>
</tr>
<tr>
<td>TILDE SUBSTITUTION</td>
<td>111</td>
</tr>
<tr>
<td>PORTABILITY ISSUES</td>
<td>111</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>111</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>112</td>
</tr>
<tr>
<td>flush</td>
<td>113</td>
</tr>
<tr>
<td>NAME</td>
<td>113</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>113</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>113</td>
</tr>
<tr>
<td>EXAMPLE</td>
<td>113</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>113</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>113</td>
</tr>
<tr>
<td>for</td>
<td>114</td>
</tr>
<tr>
<td>NAME</td>
<td>114</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>114</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>114</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>114</td>
</tr>
</tbody>
</table>
Table of Contents

Tcl Commands

SEE ALSO ...............................................................................................................................................115
KEYWORDS ...........................................................................................................................................115

foreach ..................................................................................................................................................116
NAME ....................................................................................................................................................116
SYNOPSIS .............................................................................................................................................116
DESCRIPTION .......................................................................................................................................116
EXAMPLES ...........................................................................................................................................116
SEE ALSO ...............................................................................................................................................117
KEYWORDS .............................................................................................................................................117

format ...................................................................................................................................................118
NAME ....................................................................................................................................................118
SYNOPSIS ................................................................................................................................................118
DESCRIPTION .........................................................................................................................................118
EXAMPLES ...........................................................................................................................................119
SEE ALSO ...............................................................................................................................................122
KEYWORDS .............................................................................................................................................122

gets .......................................................................................................................................................123
NAME ....................................................................................................................................................123
SYNOPSIS ................................................................................................................................................123
DESCRIPTION .........................................................................................................................................123
EXAMPLE ...............................................................................................................................................123
SEE ALSO ...............................................................................................................................................124
KEYWORDS .............................................................................................................................................124

glob .......................................................................................................................................................125
NAME ....................................................................................................................................................125
SYNOPSIS ................................................................................................................................................125
DESCRIPTION .........................................................................................................................................125
PORTABILITY ISSUES ..........................................................................................................................127
EXAMPLES ...........................................................................................................................................128
SEE ALSO ...............................................................................................................................................128
KEYWORDS .............................................................................................................................................128

global .....................................................................................................................................................129
NAME ....................................................................................................................................................129
SYNOPSIS ................................................................................................................................................129
DESCRIPTION .........................................................................................................................................129
EXAMPLES ...........................................................................................................................................129
SEE ALSO ...............................................................................................................................................129
KEYWORDS .............................................................................................................................................129

history .....................................................................................................................................................131
NAME ....................................................................................................................................................131
# Table of Contents

**Tcl Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>interp</td>
<td>SYNOPSIS</td>
<td>153</td>
</tr>
<tr>
<td>interp</td>
<td>DESCRIPTION</td>
<td>131</td>
</tr>
<tr>
<td>interp</td>
<td>HISTORY REVISION</td>
<td>132</td>
</tr>
<tr>
<td>interp</td>
<td>KEYWORDS</td>
<td>133</td>
</tr>
<tr>
<td>http</td>
<td>NAME</td>
<td>134</td>
</tr>
<tr>
<td>http</td>
<td>SYNOPSIS</td>
<td>135</td>
</tr>
<tr>
<td>http</td>
<td>DESCRIPTION</td>
<td>135</td>
</tr>
<tr>
<td>http</td>
<td>COMMANDS</td>
<td>136</td>
</tr>
<tr>
<td>http</td>
<td>ERRORS</td>
<td>140</td>
</tr>
<tr>
<td>http</td>
<td>STATE ARRAY</td>
<td>141</td>
</tr>
<tr>
<td>http</td>
<td>EXAMPLE</td>
<td>142</td>
</tr>
<tr>
<td>http</td>
<td>SEE ALSO</td>
<td>143</td>
</tr>
<tr>
<td>http</td>
<td>KEYWORDS</td>
<td>143</td>
</tr>
<tr>
<td>if</td>
<td>NAME</td>
<td>144</td>
</tr>
<tr>
<td>if</td>
<td>SYNOPSIS</td>
<td>144</td>
</tr>
<tr>
<td>if</td>
<td>DESCRIPTION</td>
<td>144</td>
</tr>
<tr>
<td>if</td>
<td>EXAMPLES</td>
<td>144</td>
</tr>
<tr>
<td>if</td>
<td>SEE ALSO</td>
<td>145</td>
</tr>
<tr>
<td>if</td>
<td>KEYWORDS</td>
<td>145</td>
</tr>
<tr>
<td>incr</td>
<td>NAME</td>
<td>146</td>
</tr>
<tr>
<td>incr</td>
<td>SYNOPSIS</td>
<td>146</td>
</tr>
<tr>
<td>incr</td>
<td>DESCRIPTION</td>
<td>146</td>
</tr>
<tr>
<td>incr</td>
<td>EXAMPLES</td>
<td>146</td>
</tr>
<tr>
<td>incr</td>
<td>SEE ALSO</td>
<td>146</td>
</tr>
<tr>
<td>incr</td>
<td>KEYWORDS</td>
<td>146</td>
</tr>
<tr>
<td>info</td>
<td>NAME</td>
<td>148</td>
</tr>
<tr>
<td>info</td>
<td>SYNOPSIS</td>
<td>148</td>
</tr>
<tr>
<td>info</td>
<td>DESCRIPTION</td>
<td>149</td>
</tr>
<tr>
<td>info</td>
<td>EXAMPLE</td>
<td>151</td>
</tr>
<tr>
<td>info</td>
<td>SEE ALSO</td>
<td>151</td>
</tr>
<tr>
<td>info</td>
<td>KEYWORDS</td>
<td>151</td>
</tr>
<tr>
<td>interp</td>
<td>NAME</td>
<td>153</td>
</tr>
<tr>
<td>interp</td>
<td>SYNOPSIS</td>
<td>154</td>
</tr>
<tr>
<td>interp</td>
<td>DESCRIPTION</td>
<td>154</td>
</tr>
<tr>
<td>interp</td>
<td>THE INTERP COMMAND</td>
<td>155</td>
</tr>
<tr>
<td>interp</td>
<td>SLAVE COMMAND</td>
<td>157</td>
</tr>
<tr>
<td>interp</td>
<td>SAFE INTERPRETERS</td>
<td>159</td>
</tr>
</tbody>
</table>
# Table of Contents

## Tcl Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALIAS INVOCATION</td>
<td>161</td>
</tr>
<tr>
<td>HIDDEN COMMANDS</td>
<td>161</td>
</tr>
<tr>
<td>CREDITS</td>
<td>161</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>162</td>
</tr>
<tr>
<td>NAME</td>
<td>164</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>164</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>164</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>164</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>164</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>164</td>
</tr>
<tr>
<td>lappend</td>
<td>165</td>
</tr>
<tr>
<td>NAME</td>
<td>165</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>165</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>165</td>
</tr>
<tr>
<td>EXAMPLE</td>
<td>165</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>165</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>165</td>
</tr>
<tr>
<td>library</td>
<td>166</td>
</tr>
<tr>
<td>NAME</td>
<td>167</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>167</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>167</td>
</tr>
<tr>
<td>COMMAND PROCEDURES</td>
<td>168</td>
</tr>
<tr>
<td>VARIABLES</td>
<td>170</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>171</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>171</td>
</tr>
<tr>
<td>lindex</td>
<td>172</td>
</tr>
<tr>
<td>NAME</td>
<td>172</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>172</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>172</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>173</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>173</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>173</td>
</tr>
<tr>
<td>linsert</td>
<td>174</td>
</tr>
<tr>
<td>NAME</td>
<td>174</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>174</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>174</td>
</tr>
<tr>
<td>EXAMPLE</td>
<td>174</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>174</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>174</td>
</tr>
</tbody>
</table>
Table of Contents

Tcl Commands

list........................................................................................................................................................................175
NAME......................................................................................................................................................................175
SYNOPSIS..................................................................................................................................................................175
DESCRIPTION............................................................................................................................................................175
EXAMPLE....................................................................................................................................................................175
SEE ALSO ....................................................................................................................................................................175
KEYWORDS ..................................................................................................................................................................175

llength.........................................................................................................................................................................176
NAME.........................................................................................................................................................................176
SYNOPSIS.....................................................................................................................................................................176
DESCRIPTION..................................................................................................................................................................176
EXAMPLES..................................................................................................................................................................176
SEE ALSO....................................................................................................................................................................176
KEYWORDS ..................................................................................................................................................................176

load.............................................................................................................................................................................178
NAME.........................................................................................................................................................................178
SYNOPSIS.....................................................................................................................................................................178
DESCRIPTION..................................................................................................................................................................178
PORTABILITY ISSUES ................................................................................................................................................179
BUGS..............................................................................................................................................................................180
EXAMPLE....................................................................................................................................................................180
SEE ALSO....................................................................................................................................................................180
KEYWORDS ..................................................................................................................................................................181

lrange...........................................................................................................................................................................182
NAME.........................................................................................................................................................................182
SYNOPSIS.....................................................................................................................................................................182
DESCRIPTION..................................................................................................................................................................182
EXAMPLES..................................................................................................................................................................182
SEE ALSO....................................................................................................................................................................183
KEYWORDS ..................................................................................................................................................................183

lreplace.......................................................................................................................................................................184
NAME.........................................................................................................................................................................184
SYNOPSIS.....................................................................................................................................................................184
DESCRIPTION..................................................................................................................................................................184
EXAMPLES..................................................................................................................................................................184
SEE ALSO....................................................................................................................................................................184
KEYWORDS ..................................................................................................................................................................185

lsearch..........................................................................................................................................................................186
NAME.........................................................................................................................................................................186
SYNOPSIS.....................................................................................................................................................................186
DESCRIPTION..................................................................................................................................................................186
EXAMPLES..................................................................................................................................................................186
**Tcl Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>See Also</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>namespace</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>msgcat</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>memory</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>lsort</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>lset</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**WHAT IS A NAMESPACE?**

**DESCRIPTION**

**SYNOPSIS**

**NAME**

**KEYWORDS**

**SEE ALSO**

**EXAMPLES**

**NOTES**

**LOCATION AND FORMAT OF MESSAGE FILES**

**RECOMMENDED MESSAGE SETUP FOR PACKAGES**

**POSITIONAL CODES FOR FORMAT AND SCAN COMMANDS**

**CREDITS**

**SEE ALSO**

**EXAMPLES**

**DESCRIPTION**

**SYNOPSIS**

**NAME**

**KEYWORDS**

**SEE ALSO**

**EXAMPLES**

**DESCRIPTION**

**SYNOPSIS**

**NAME**

**KEYWORDS**

**SEE ALSO**

**EXAMPLES**

**DESCRIPTION**

**SYNOPSIS**

**NAME**

**KEYWORDS**

**SEE ALSO**

**EXAMPLES**

**DESCRIPTION**

**SYNOPSIS**

**NAME**

**KEYWORDS**

**SEE ALSO**

**EXAMPLES**

**DESCRIPTION**

**SYNOPSIS**

**NAME**

**KEYWORDS**

**SEE ALSO**

**EXAMPLES**
# Table of Contents

## Tcl Commands

### NAME RESOLUTION
- START
- END

### IMPORTING COMMANDS
- START
- END

### EXPORTING COMMANDS
- START
- END

### SCOPED SCRIPTS
- START
- END

### EXAMPLES
- START
- END

### SEE ALSO
- START
- END

### OPTIONS
- START
- END

### SUMMARY
- START
- END

### NAME
- START
- END

### SYNOPSIS
- START
- END

### DESCRIPTION
- START
- END

### ERROR CODES (Windows only)
- START
- END

### SERIAL PORT SIGNALS
- START
- END

### SERIAL COMMUNICATIONS
- START
- END

### PORTABILITY ISSUES
- START
- END

### COMMAND PIPELINES
- START
- END

### SCOPED SCRIPTS
- START
- END

### EXPORTING COMMANDS
- START
- END

### IMPORTING COMMANDS
- START
- END

### NAME RESOLUTION
- START
- END

### VERSION NUMBERS
- START
- END

### PACKAGE INDICES
- START
- END

### EXAMPLES
- START
- END

### SEE ALSO
- START
- END

### KEYWORDS
- START
- END

### SERIAL COMMUNICATIONS
- START
- END

### COMMAND PIPELINES
- START
- END

### ERROR CODES (Windows only)
- START
- END

### SERIAL PORT SIGNALS
- START
- END

### COMMAND PIPELINES
- START
- END

### SCOPED SCRIPTS
- START
- END

### EXPORTING COMMANDS
- START
- END

### IMPORTING COMMANDS
- START
- END

### NAME RESOLUTION
- START
- END

### VERSION NUMBERS
- START
- END

### PACKAGE INDICES
- START
- END

### EXAMPLES
- START
- END

### SEE ALSO
- START
- END

### KEYWORDS
- START
- END
# Table of Contents

## Tcl Commands

**pkgMkIndex**

- **NAME**
- **SYNOPSIS**
- **DESCRIPTION**
- **OPTIONS**
- **PACKAGES AND THE AUTO-LOADER**
- **HOW IT WORKS**
- **DIRECT LOADING**
- **COMPLEX CASES**
- **SEE ALSO**
- **KEYWORDS**

**proc**

- **NAME**
- **SYNOPSIS**
- **DESCRIPTION**
- **EXAMPLES**
- **SEE ALSO**
- **KEYWORDS**

**puts**

- **NAME**
- **SYNOPSIS**
- **DESCRIPTION**
- **EXAMPLES**
- **SEE ALSO**
- **KEYWORDS**

**pwd**

- **NAME**
- **SYNOPSIS**
- **DESCRIPTION**
- **EXAMPLE**
- **SEE ALSO**
- **KEYWORDS**

**re_syntax**

- **NAME**
- **DESCRIPTION**
- **DIFFERENT FLAVORS OF REs**
- **REGULAR EXPRESSION SYNTAX**
- **BRACKET EXPRESSIONS**
- **ESCAPES**
- **METHALS**
- **MATCHING**
- **LIMITS AND COMPATIBILITY**
# Table of Contents

## Tcl Commands

**BASIC REGULAR EXPRESSIONS** ........................................................................................................... 253
**SEE ALSO** .................................................................................................................................................. 253
**KEYWORDS** ........................................................................................................................................... 253
**read** ........................................................................................................................................................................... 254
**NAME** ................................................................................................................................................................. 254
**SYNOPSIS** ...................................................................................................................................................... 254
**DESCRIPTION** ............................................................................................................................................... 254
**USE WITH SERIAL PORTS** ................................................................................................................................. 255
**EXAMPLE** .................................................................................................................................................... 255
**SEE ALSO** .................................................................................................................................................... 255
**KEYWORDS** .................................................................................................................................................... 255
**regexp** ............................................................................................................................................................... 256
**NAME** ............................................................................................................................................................... 256
**SYNOPSIS** ........................................................................................................................................................ 256
**DESCRIPTION** ............................................................................................................................................... 256
**EXAMPLES** .................................................................................................................................................... 258
**SEE ALSO** .................................................................................................................................................... 258
**KEYWORDS** .................................................................................................................................................... 258
**registry** ............................................................................................................................................................. 259
**NAME** ............................................................................................................................................................. 259
**SYNOPSIS** ........................................................................................................................................................ 259
**DESCRIPTION** ............................................................................................................................................... 259
**SUPPORTED TYPES** ........................................................................................................................................... 261
**PORTABILITY ISSUES** ....................................................................................................................................... 262
**EXAMPLE** .................................................................................................................................................... 262
**KEYWORDS** .................................................................................................................................................... 262
**regsub** ............................................................................................................................................................... 263
**NAME** ............................................................................................................................................................... 263
**SYNOPSIS** ........................................................................................................................................................ 263
**DESCRIPTION** ............................................................................................................................................... 263
**EXAMPLES** .................................................................................................................................................... 264
**SEE ALSO** .................................................................................................................................................... 265
**KEYWORDS** .................................................................................................................................................... 265
**rename** ............................................................................................................................................................. 266
**NAME** ............................................................................................................................................................. 266
**SYNOPSIS** ........................................................................................................................................................ 266
**DESCRIPTION** ............................................................................................................................................... 266
**EXAMPLE** .................................................................................................................................................... 266
**SEE ALSO** .................................................................................................................................................... 266
**KEYWORDS** .................................................................................................................................................... 266
**resource** ............................................................................................................................................................. 267
**NAME** ............................................................................................................................................................. 267
# Table of Contents

## Tcl Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>set</td>
<td>281</td>
</tr>
<tr>
<td>seek</td>
<td>279</td>
</tr>
<tr>
<td>scan</td>
<td>274</td>
</tr>
<tr>
<td>return</td>
<td>271</td>
</tr>
</tbody>
</table>

## Tcl8.4/Tk8.4 Manual

---

xv
## Table of Contents

### Tcl Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLIENT SOCKETS</td>
<td>284</td>
</tr>
<tr>
<td>SERVER SOCKETS</td>
<td>284</td>
</tr>
<tr>
<td>CONFIGURATION OPTIONS</td>
<td>285</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>285</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>286</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>286</td>
</tr>
<tr>
<td>source</td>
<td>287</td>
</tr>
<tr>
<td>NAME</td>
<td>287</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>287</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>287</td>
</tr>
<tr>
<td>EXAMPLE</td>
<td>287</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>288</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>288</td>
</tr>
<tr>
<td>split</td>
<td>289</td>
</tr>
<tr>
<td>NAME</td>
<td>289</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>289</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>289</td>
</tr>
<tr>
<td>EXAMPLE</td>
<td>289</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>290</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>290</td>
</tr>
<tr>
<td>string</td>
<td>291</td>
</tr>
<tr>
<td>NAME</td>
<td>292</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>292</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>292</td>
</tr>
<tr>
<td>EXAMPLE</td>
<td>297</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>297</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>297</td>
</tr>
<tr>
<td>subst</td>
<td>298</td>
</tr>
<tr>
<td>NAME</td>
<td>298</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>298</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>298</td>
</tr>
<tr>
<td>EXAMPLE</td>
<td>298</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>300</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>300</td>
</tr>
<tr>
<td>switch</td>
<td>301</td>
</tr>
<tr>
<td>NAME</td>
<td>301</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>301</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>301</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>302</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>303</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>303</td>
</tr>
</tbody>
</table>
# Table of Contents

## Tcl Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcltest</td>
<td>304</td>
</tr>
<tr>
<td></td>
<td>NAME</td>
</tr>
<tr>
<td></td>
<td>SYNOPSIS</td>
</tr>
<tr>
<td></td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td></td>
<td>COMMANDS</td>
</tr>
<tr>
<td></td>
<td>CONFIGURATION COMMANDS</td>
</tr>
<tr>
<td></td>
<td>SHORTCUT COMMANDS</td>
</tr>
<tr>
<td></td>
<td>OTHER COMMANDS</td>
</tr>
<tr>
<td></td>
<td>TESTS</td>
</tr>
<tr>
<td></td>
<td>TEST CONSTRAINTS</td>
</tr>
<tr>
<td></td>
<td>RUNNING ALL TESTS</td>
</tr>
<tr>
<td></td>
<td>CONFIGURABLE OPTIONS</td>
</tr>
<tr>
<td></td>
<td>CREATING TEST SUITES WITH TCLTEST</td>
</tr>
<tr>
<td></td>
<td>COMPATIBILITY</td>
</tr>
<tr>
<td></td>
<td>KNOWN ISSUES</td>
</tr>
<tr>
<td></td>
<td>KEYWORDS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>tclvars</td>
<td>324</td>
</tr>
<tr>
<td></td>
<td>NAME</td>
</tr>
<tr>
<td></td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td></td>
<td>OTHER GLOBAL VARIABLES</td>
</tr>
<tr>
<td></td>
<td>SEE ALSO</td>
</tr>
<tr>
<td></td>
<td>KEYWORDS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>tell</td>
<td>332</td>
</tr>
<tr>
<td></td>
<td>NAME</td>
</tr>
<tr>
<td></td>
<td>SYNOPSIS</td>
</tr>
<tr>
<td></td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td></td>
<td>EXAMPLE</td>
</tr>
<tr>
<td></td>
<td>SEE ALSO</td>
</tr>
<tr>
<td></td>
<td>KEYWORDS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>333</td>
</tr>
<tr>
<td></td>
<td>NAME</td>
</tr>
<tr>
<td></td>
<td>SYNOPSIS</td>
</tr>
<tr>
<td></td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td></td>
<td>EXAMPLE</td>
</tr>
<tr>
<td></td>
<td>SEE ALSO</td>
</tr>
<tr>
<td></td>
<td>KEYWORDS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>trace</td>
<td>334</td>
</tr>
<tr>
<td></td>
<td>NAME</td>
</tr>
<tr>
<td></td>
<td>SYNOPSIS</td>
</tr>
<tr>
<td></td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td></td>
<td>EXAMPLES</td>
</tr>
<tr>
<td></td>
<td>SEE ALSO</td>
</tr>
</tbody>
</table>

## Other Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTHER COMMANDS</td>
<td>312</td>
</tr>
</tbody>
</table>

## Known Issues

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNOWN ISSUES</td>
<td>323</td>
</tr>
</tbody>
</table>

## Other Global Variables

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTHER GLOBAL VARIABLES</td>
<td>330</td>
</tr>
</tbody>
</table>

## Configuration

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONFIGURATION COMMANDS</td>
<td>310</td>
</tr>
<tr>
<td>SHORTCUT COMMANDS</td>
<td>311</td>
</tr>
<tr>
<td>OTHER COMMANDS</td>
<td>312</td>
</tr>
<tr>
<td>TESTS</td>
<td>313</td>
</tr>
<tr>
<td>TEST CONSTRAINTS</td>
<td>315</td>
</tr>
<tr>
<td>RUNNING ALL TESTS</td>
<td>317</td>
</tr>
<tr>
<td>CONFIGURABLE OPTIONS</td>
<td>318</td>
</tr>
<tr>
<td>CREATING TEST SUITES WITH TCLTEST</td>
<td>320</td>
</tr>
<tr>
<td>COMPATIBILITY</td>
<td>322</td>
</tr>
<tr>
<td>KNOWN ISSUES</td>
<td>323</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>323</td>
</tr>
</tbody>
</table>

## Tcl8.4/Tk8.4 Manual

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONFIGURABLE OPTIONS</td>
<td>318</td>
</tr>
<tr>
<td>TESTING</td>
<td>315</td>
</tr>
<tr>
<td>RUNNING ALL TESTS</td>
<td>317</td>
</tr>
<tr>
<td>CONFIGURATION</td>
<td>312</td>
</tr>
<tr>
<td>OTHER COMMANDS</td>
<td>312</td>
</tr>
<tr>
<td>TESTS</td>
<td>313</td>
</tr>
<tr>
<td>TEST CONSTRAINTS</td>
<td>315</td>
</tr>
<tr>
<td>RUNNING ALL TESTS</td>
<td>317</td>
</tr>
<tr>
<td>CONFIGURABLE OPTIONS</td>
<td>318</td>
</tr>
<tr>
<td>CREATING TEST SUITES WITH TCLTEST</td>
<td>320</td>
</tr>
<tr>
<td>COMPATIBILITY</td>
<td>322</td>
</tr>
<tr>
<td>KNOWN ISSUES</td>
<td>323</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>323</td>
</tr>
<tr>
<td>OTHER COMMANDS</td>
<td>312</td>
</tr>
<tr>
<td>TESTS</td>
<td>313</td>
</tr>
<tr>
<td>TEST CONSTRAINTS</td>
<td>315</td>
</tr>
<tr>
<td>RUNNING ALL TESTS</td>
<td>317</td>
</tr>
<tr>
<td>CONFIGURABLE OPTIONS</td>
<td>318</td>
</tr>
<tr>
<td>CREATING TEST SUITES WITH TCLTEST</td>
<td>320</td>
</tr>
<tr>
<td>COMPATIBILITY</td>
<td>322</td>
</tr>
<tr>
<td>KNOWN ISSUES</td>
<td>323</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>323</td>
</tr>
<tr>
<td>OTHER COMMANDS</td>
<td>312</td>
</tr>
<tr>
<td>TESTS</td>
<td>313</td>
</tr>
<tr>
<td>TEST CONSTRAINTS</td>
<td>315</td>
</tr>
<tr>
<td>RUNNING ALL TESTS</td>
<td>317</td>
</tr>
<tr>
<td>CONFIGURABLE OPTIONS</td>
<td>318</td>
</tr>
<tr>
<td>CREATING TEST SUITES WITH TCLTEST</td>
<td>320</td>
</tr>
<tr>
<td>COMPATIBILITY</td>
<td>322</td>
</tr>
<tr>
<td>KNOWN ISSUES</td>
<td>323</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>323</td>
</tr>
</tbody>
</table>

## Table of Contents

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcltest</td>
<td>304</td>
</tr>
<tr>
<td></td>
<td>NAME</td>
</tr>
<tr>
<td></td>
<td>SYNOPSIS</td>
</tr>
<tr>
<td></td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td></td>
<td>COMMANDS</td>
</tr>
<tr>
<td></td>
<td>CONFIGURATION COMMANDS</td>
</tr>
<tr>
<td></td>
<td>SHORTCUT COMMANDS</td>
</tr>
<tr>
<td></td>
<td>OTHER COMMANDS</td>
</tr>
<tr>
<td></td>
<td>TESTS</td>
</tr>
<tr>
<td></td>
<td>TEST CONSTRAINTS</td>
</tr>
<tr>
<td></td>
<td>RUNNING ALL TESTS</td>
</tr>
<tr>
<td></td>
<td>CONFIGURABLE OPTIONS</td>
</tr>
<tr>
<td></td>
<td>CREATING TEST SUITES WITH TCLTEST</td>
</tr>
<tr>
<td></td>
<td>COMPATIBILITY</td>
</tr>
<tr>
<td></td>
<td>KNOWN ISSUES</td>
</tr>
<tr>
<td></td>
<td>KEYWORDS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>tclvars</td>
<td>324</td>
</tr>
<tr>
<td></td>
<td>NAME</td>
</tr>
<tr>
<td></td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td></td>
<td>OTHER GLOBAL VARIABLES</td>
</tr>
<tr>
<td></td>
<td>SEE ALSO</td>
</tr>
<tr>
<td></td>
<td>KEYWORDS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>tell</td>
<td>332</td>
</tr>
<tr>
<td></td>
<td>NAME</td>
</tr>
<tr>
<td></td>
<td>SYNOPSIS</td>
</tr>
<tr>
<td></td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td></td>
<td>EXAMPLE</td>
</tr>
<tr>
<td></td>
<td>SEE ALSO</td>
</tr>
<tr>
<td></td>
<td>KEYWORDS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>333</td>
</tr>
<tr>
<td></td>
<td>NAME</td>
</tr>
<tr>
<td></td>
<td>SYNOPSIS</td>
</tr>
<tr>
<td></td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td></td>
<td>EXAMPLE</td>
</tr>
<tr>
<td></td>
<td>SEE ALSO</td>
</tr>
<tr>
<td></td>
<td>KEYWORDS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>trace</td>
<td>334</td>
</tr>
<tr>
<td></td>
<td>NAME</td>
</tr>
<tr>
<td></td>
<td>SYNOPSIS</td>
</tr>
<tr>
<td></td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td></td>
<td>EXAMPLES</td>
</tr>
<tr>
<td></td>
<td>SEE ALSO</td>
</tr>
</tbody>
</table>
# Table of Contents

## Tcl Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Synopsis</th>
<th>Example</th>
<th>See Also</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>unset</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>update</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>uplevel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>upvar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>variable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unknown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>setc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**KEYWORDS**

- unset
- update
- uplevel
- upvar
- variable
- unknown
- setc

**NAME**

- unset
- update
- uplevel
- upvar
- variable
- unknown
- setc

**SYNOPSIS**

- unset
- update
- uplevel
- upvar
- variable
- unknown
- setc

**DESCRIPTION**

- unset
- update
- uplevel
- upvar
- variable
- unknown
- setc

**EXAMPLE**

- unset
- update
- uplevel
- upvar
- variable
- unknown
- setc

**SEE ALSO**

- unset
- update
- uplevel
- upvar
- variable
- unknown
- setc

**KEYWORDS**

- unset
- update
- uplevel
- upvar
- variable
- unknown
- setc

---

**Tcl8.4/Tk8.4 Manual**

xviii
# Table of Contents

**Tk Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Synopses</th>
<th>Description</th>
<th>Examples</th>
<th>See Also</th>
</tr>
</thead>
<tbody>
<tr>
<td>while</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vwait</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Tk Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Synopses</th>
<th>Description</th>
<th>Examples</th>
<th>See Also</th>
</tr>
</thead>
<tbody>
<tr>
<td>loadTk</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bell</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bind</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Tcl Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Synopses</th>
<th>Description</th>
<th>Examples</th>
<th>See Also</th>
</tr>
</thead>
<tbody>
<tr>
<td>bell</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>while</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vwait</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Table of Contents

## Tk Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXAMPLES</td>
<td>373</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>373</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>373</td>
</tr>
<tr>
<td>bindtags</td>
<td>374</td>
</tr>
<tr>
<td>NAME</td>
<td>374</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>374</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>374</td>
</tr>
<tr>
<td>EXAMPLE</td>
<td>375</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>375</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>375</td>
</tr>
<tr>
<td>bitmap</td>
<td>376</td>
</tr>
<tr>
<td>NAME</td>
<td>376</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>376</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>376</td>
</tr>
<tr>
<td>CREATING BITMAPS</td>
<td>376</td>
</tr>
<tr>
<td>IMAGE COMMAND</td>
<td>377</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>378</td>
</tr>
<tr>
<td>button</td>
<td>379</td>
</tr>
<tr>
<td>NAME</td>
<td>380</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>380</td>
</tr>
<tr>
<td>STANDARD OPTIONS</td>
<td>380</td>
</tr>
<tr>
<td>WIDGET–SPECIFIC OPTIONS</td>
<td>381</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>382</td>
</tr>
<tr>
<td>WIDGET COMMAND</td>
<td>382</td>
</tr>
<tr>
<td>DEFAULT BINDINGS</td>
<td>383</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>383</td>
</tr>
<tr>
<td>canvas</td>
<td>384</td>
</tr>
<tr>
<td>NAME</td>
<td>388</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>388</td>
</tr>
<tr>
<td>STANDARD OPTIONS</td>
<td>388</td>
</tr>
<tr>
<td>WIDGET–SPECIFIC OPTIONS</td>
<td>388</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>390</td>
</tr>
<tr>
<td>DISPLAY LIST</td>
<td>390</td>
</tr>
<tr>
<td>ITEM IDS AND TAGS</td>
<td>390</td>
</tr>
<tr>
<td>COORDINATES</td>
<td>391</td>
</tr>
<tr>
<td>TRANSFORMATIONS</td>
<td>391</td>
</tr>
<tr>
<td>INDICES</td>
<td>392</td>
</tr>
<tr>
<td>DASH PATTERNS</td>
<td>393</td>
</tr>
<tr>
<td>WIDGET COMMAND</td>
<td>393</td>
</tr>
<tr>
<td>OVERVIEW OF ITEM TYPES</td>
<td>402</td>
</tr>
<tr>
<td>COMMON ITEM OPTIONS</td>
<td>402</td>
</tr>
<tr>
<td>ARC ITEMS</td>
<td>404</td>
</tr>
</tbody>
</table>
Table of Contents

Tk Commands

BITMAP ITEMS ....................................................................................................................................................405
IMAGE ITEMS .......................................................................................................................................................406
LINE ITEMS .........................................................................................................................................................406
OVAL ITEMS ........................................................................................................................................................408
POLYGON ITEMS ...............................................................................................................................................408
RECTANGLE ITEMS ........................................................................................................................................410
TEXT ITEMS .......................................................................................................................................................410
WINDOW ITEMS ..................................................................................................................................................412
APPLICATION–DEFINED ITEM TYPES ........................................................................................................412
BINDINGS ..........................................................................................................................................................413
CREDITS ............................................................................................................................................................413
SEE ALSO ..........................................................................................................................................................420
KEYWORDS .......................................................................................................................................................420

checkbutton ......................................................................................................................................................414

NAME ...............................................................................................................................................................415
SYNOPSIS ..........................................................................................................................................................415
STANDARD OPTIONS ......................................................................................................................................415
WIDGET–SPECIFIC OPTIONS ..........................................................................................................................416
DESCRIPTION ..................................................................................................................................................417
WIDGET COMMAND .........................................................................................................................................418
BINDINGS .........................................................................................................................................................419
SEE ALSO ..........................................................................................................................................................420
KEYWORDS ..........................................................................................................................................................420

clipboard .............................................................................................................................................................421

NAME ...............................................................................................................................................................421
SYNOPSIS ..........................................................................................................................................................421
DESCRIPTION ..................................................................................................................................................421
EXAMPLES .......................................................................................................................................................422
SEE ALSO ..........................................................................................................................................................422
KEYWORDS ..........................................................................................................................................................422

colors .................................................................................................................................................................423

NAME ...............................................................................................................................................................423
DESCRIPTION ..................................................................................................................................................423
KEYWORDS .......................................................................................................................................................423

console .................................................................................................................................................................438

NAME ...............................................................................................................................................................438
SYNOPSIS ..........................................................................................................................................................438
DESCRIPTION ..................................................................................................................................................438
ACCESS TO THE MAIN INTERPRETER ........................................................................................................439
ADDITIONAL TRAP CALLS ................................................................................................................................439
DEFAULT BINDINGS .........................................................................................................................................439
EXAMPLE ..........................................................................................................................................................440
# Table of Contents

## Tk Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursors</td>
<td>441</td>
</tr>
<tr>
<td>destroy</td>
<td>443</td>
</tr>
<tr>
<td>entry</td>
<td>444</td>
</tr>
<tr>
<td>event</td>
<td>456</td>
</tr>
<tr>
<td>focus</td>
<td>463</td>
</tr>
<tr>
<td>font</td>
<td>467</td>
</tr>
</tbody>
</table>

## NAME

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursors</td>
<td>441</td>
</tr>
<tr>
<td>destroy</td>
<td>444</td>
</tr>
<tr>
<td>entry</td>
<td>447</td>
</tr>
<tr>
<td>event</td>
<td>458</td>
</tr>
<tr>
<td>focus</td>
<td>464</td>
</tr>
<tr>
<td>font</td>
<td>468</td>
</tr>
</tbody>
</table>

## DESCRIPTION

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursors</td>
<td>441</td>
</tr>
<tr>
<td>destroy</td>
<td>444</td>
</tr>
<tr>
<td>entry</td>
<td>447</td>
</tr>
<tr>
<td>event</td>
<td>458</td>
</tr>
<tr>
<td>focus</td>
<td>464</td>
</tr>
<tr>
<td>font</td>
<td>468</td>
</tr>
</tbody>
</table>

## SYNONYMS

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursors</td>
<td>441</td>
</tr>
<tr>
<td>destroy</td>
<td>444</td>
</tr>
<tr>
<td>entry</td>
<td>447</td>
</tr>
<tr>
<td>event</td>
<td>458</td>
</tr>
<tr>
<td>focus</td>
<td>464</td>
</tr>
<tr>
<td>font</td>
<td>468</td>
</tr>
</tbody>
</table>

## KEYWORDS

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursors</td>
<td>441</td>
</tr>
<tr>
<td>destroy</td>
<td>444</td>
</tr>
<tr>
<td>entry</td>
<td>447</td>
</tr>
<tr>
<td>event</td>
<td>458</td>
</tr>
<tr>
<td>focus</td>
<td>464</td>
</tr>
<tr>
<td>font</td>
<td>468</td>
</tr>
</tbody>
</table>

## SEE ALSO

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursors</td>
<td>441</td>
</tr>
<tr>
<td>destroy</td>
<td>444</td>
</tr>
<tr>
<td>entry</td>
<td>447</td>
</tr>
<tr>
<td>event</td>
<td>458</td>
</tr>
<tr>
<td>focus</td>
<td>464</td>
</tr>
<tr>
<td>font</td>
<td>468</td>
</tr>
</tbody>
</table>

## VIRTUAL EVENT EXAMPLES

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursors</td>
<td>441</td>
</tr>
<tr>
<td>destroy</td>
<td>444</td>
</tr>
<tr>
<td>entry</td>
<td>447</td>
</tr>
<tr>
<td>event</td>
<td>458</td>
</tr>
<tr>
<td>focus</td>
<td>464</td>
</tr>
<tr>
<td>font</td>
<td>468</td>
</tr>
</tbody>
</table>

## EVENT FIELDS

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursors</td>
<td>441</td>
</tr>
<tr>
<td>destroy</td>
<td>444</td>
</tr>
<tr>
<td>entry</td>
<td>447</td>
</tr>
<tr>
<td>event</td>
<td>458</td>
</tr>
<tr>
<td>focus</td>
<td>464</td>
</tr>
<tr>
<td>font</td>
<td>468</td>
</tr>
</tbody>
</table>

## DEFAULT BINDINGS

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursors</td>
<td>441</td>
</tr>
<tr>
<td>destroy</td>
<td>444</td>
</tr>
<tr>
<td>entry</td>
<td>447</td>
</tr>
<tr>
<td>event</td>
<td>458</td>
</tr>
<tr>
<td>focus</td>
<td>464</td>
</tr>
<tr>
<td>font</td>
<td>468</td>
</tr>
</tbody>
</table>

## STANDARD OPTIONS

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursors</td>
<td>441</td>
</tr>
<tr>
<td>destroy</td>
<td>444</td>
</tr>
<tr>
<td>entry</td>
<td>447</td>
</tr>
<tr>
<td>event</td>
<td>458</td>
</tr>
<tr>
<td>focus</td>
<td>464</td>
</tr>
<tr>
<td>font</td>
<td>468</td>
</tr>
</tbody>
</table>

## PORTABILITY ISSUES

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursors</td>
<td>441</td>
</tr>
<tr>
<td>destroy</td>
<td>444</td>
</tr>
<tr>
<td>entry</td>
<td>447</td>
</tr>
<tr>
<td>event</td>
<td>458</td>
</tr>
<tr>
<td>focus</td>
<td>464</td>
</tr>
<tr>
<td>font</td>
<td>468</td>
</tr>
</tbody>
</table>

## VALIDATION

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursors</td>
<td>441</td>
</tr>
<tr>
<td>destroy</td>
<td>444</td>
</tr>
<tr>
<td>entry</td>
<td>447</td>
</tr>
<tr>
<td>event</td>
<td>458</td>
</tr>
<tr>
<td>focus</td>
<td>464</td>
</tr>
<tr>
<td>font</td>
<td>468</td>
</tr>
</tbody>
</table>

## WIDGET SPECIFIC OPTIONS

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursors</td>
<td>441</td>
</tr>
<tr>
<td>destroy</td>
<td>444</td>
</tr>
<tr>
<td>entry</td>
<td>447</td>
</tr>
<tr>
<td>event</td>
<td>458</td>
</tr>
<tr>
<td>focus</td>
<td>464</td>
</tr>
<tr>
<td>font</td>
<td>468</td>
</tr>
</tbody>
</table>

## STANDARD OPTIONS

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursors</td>
<td>441</td>
</tr>
<tr>
<td>destroy</td>
<td>444</td>
</tr>
<tr>
<td>entry</td>
<td>447</td>
</tr>
<tr>
<td>event</td>
<td>458</td>
</tr>
<tr>
<td>focus</td>
<td>464</td>
</tr>
<tr>
<td>font</td>
<td>468</td>
</tr>
</tbody>
</table>

## SEE ALSO

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursors</td>
<td>441</td>
</tr>
<tr>
<td>destroy</td>
<td>444</td>
</tr>
<tr>
<td>entry</td>
<td>447</td>
</tr>
<tr>
<td>event</td>
<td>458</td>
</tr>
<tr>
<td>focus</td>
<td>464</td>
</tr>
<tr>
<td>font</td>
<td>468</td>
</tr>
</tbody>
</table>

## QUIRKS

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursors</td>
<td>441</td>
</tr>
<tr>
<td>destroy</td>
<td>444</td>
</tr>
<tr>
<td>entry</td>
<td>447</td>
</tr>
<tr>
<td>event</td>
<td>458</td>
</tr>
<tr>
<td>focus</td>
<td>464</td>
</tr>
<tr>
<td>font</td>
<td>468</td>
</tr>
</tbody>
</table>

## DESCRIPTION

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursors</td>
<td>441</td>
</tr>
<tr>
<td>destroy</td>
<td>444</td>
</tr>
<tr>
<td>entry</td>
<td>447</td>
</tr>
<tr>
<td>event</td>
<td>458</td>
</tr>
<tr>
<td>focus</td>
<td>464</td>
</tr>
<tr>
<td>font</td>
<td>468</td>
</tr>
</tbody>
</table>
# Table of Contents

**Tk Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>FONT DESCRIPTION</td>
<td>469</td>
</tr>
<tr>
<td>FONT METRICS</td>
<td>470</td>
</tr>
<tr>
<td>FONT OPTIONS</td>
<td>470</td>
</tr>
<tr>
<td>PLATFORM–SPECIFIC ISSUES</td>
<td>471</td>
</tr>
<tr>
<td>EXAMPLE</td>
<td>472</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>472</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>472</td>
</tr>
<tr>
<td>frame</td>
<td>473</td>
</tr>
<tr>
<td>NAME</td>
<td>473</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>473</td>
</tr>
<tr>
<td>STANDARD OPTIONS</td>
<td>474</td>
</tr>
<tr>
<td>WIDGET–SPECIFIC OPTIONS</td>
<td>474</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>475</td>
</tr>
<tr>
<td>WIDGET COMMAND</td>
<td>475</td>
</tr>
<tr>
<td>BINDINGS</td>
<td>476</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>476</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>476</td>
</tr>
<tr>
<td>grab</td>
<td>477</td>
</tr>
<tr>
<td>NAME</td>
<td>477</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>477</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>477</td>
</tr>
<tr>
<td>WARNING</td>
<td>478</td>
</tr>
<tr>
<td>BUGS</td>
<td>479</td>
</tr>
<tr>
<td>EXAMPLE</td>
<td>479</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>479</td>
</tr>
<tr>
<td>grid</td>
<td>480</td>
</tr>
<tr>
<td>NAME</td>
<td>481</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>481</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>481</td>
</tr>
<tr>
<td>RELATIVE PLACEMENT</td>
<td>484</td>
</tr>
<tr>
<td>THE GRID ALGORITHM</td>
<td>484</td>
</tr>
<tr>
<td>GEOMETRY PROPAGATION</td>
<td>485</td>
</tr>
<tr>
<td>RESTRICTIONS ON MASTER WINDOWS</td>
<td>485</td>
</tr>
<tr>
<td>STACKING ORDER</td>
<td>485</td>
</tr>
<tr>
<td>CREDITS</td>
<td>486</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>486</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>486</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>486</td>
</tr>
<tr>
<td>image</td>
<td>487</td>
</tr>
<tr>
<td>NAME</td>
<td>487</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>487</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>487</td>
</tr>
</tbody>
</table>
# Table of Contents

**Tk Commands**

- **BUILT-IN IMAGE TYPES** ................................................................. 488
- **SEE ALSO** .................................................................................. 488
- **KEYWORDS** ............................................................................... 489

- **keysyms** 
  - **NAME** .................................................................................. 490
  - **DESCRIPTION** ....................................................................... 490
  - **SEE ALSO** ............................................................................. 506
  - **KEYWORDS** ........................................................................... 506

- **label** 
  - **NAME** .................................................................................. 507
  - **SYNOPSIS** ............................................................................ 508
  - **STANDARD OPTIONS** ............................................................ 508
  - **WIDGET–SPECIFIC OPTIONS** ............................................... 508
  - **DESCRIPTION** ...................................................................... 509
  - **WIDGET COMMAND** .............................................................. 509
  - **BINDINGS** ............................................................................ 510
  - **KEYWORDS** ........................................................................... 510

- **labelFrame** 
  - **NAME** .................................................................................. 511
  - **SYNOPSIS** ............................................................................ 511
  - **STANDARD OPTIONS** ............................................................. 512
  - **WIDGET–SPECIFIC OPTIONS** ................................................. 512
  - **DESCRIPTION** ...................................................................... 513
  - **WIDGET COMMAND** .............................................................. 514
  - **BINDINGS** ............................................................................ 514
  - **SEE ALSO** ............................................................................. 514
  - **KEYWORDS** ........................................................................... 514

- **listbox** 
  - **NAME** .................................................................................. 515
  - **SYNOPSIS** ............................................................................ 516
  - **STANDARD OPTIONS** ............................................................. 517
  - **WIDGET–SPECIFIC OPTIONS** ................................................. 517
  - **DESCRIPTION** ...................................................................... 518
  - **INDICES** ............................................................................... 519
  - **WIDGET COMMAND** .............................................................. 519
  - **DEFAULT BINDINGS** .............................................................. 523
  - **KEYWORDS** ........................................................................... 525

- **lower** 
  - **NAME** .................................................................................. 526
  - **SYNOPSIS** ............................................................................ 526
  - **DESCRIPTION** ...................................................................... 526
  - **SEE ALSO** ............................................................................. 526
# Table of Contents

## Tk Commands

### KEYWORDS

- menu ...................................................................................................................................................527
- NAME ...........................................................................................................................................527
- SYNOPSIS ..................................................................................................................................527
- STANDARD OPTIONS ..................................................................................................................528
- WIDGET–SPECIFIC OPTIONS ........................................................................................................528
- INTRODUCTION ..........................................................................................................................530
- COMMAND ENTRY E N T R I E S ..................................................................................................531
- SEPARATOR ENTRY E N T R I E S ..................................................................................................531
- CHECKBUTTON ENTRY E N T R I E S ............................................................................................531
- RADIOBUTTON ENTRY E N T R I E S ............................................................................................531
- CASCADE ENTRY E N T R I E S ......................................................................................................532
- TEAR−OFF ENTRY E N T R I E S ......................................................................................................532
- MENUBARS ...................................................................................................................................533
- SPECIAL MENUS IN MENUBARS ...............................................................................................533
- CLONES .........................................................................................................................................534
- WIDGET COMMAND .....................................................................................................................534
- MENU CONFIGURATIONS ..............................................................................................................538
- DEFAULT BINDINGS .......................................................................................................................540
- BUGS ..............................................................................................................................................540
- KEYWORDS ....................................................................................................................................540

### menubutton

- NAME ...........................................................................................................................................541
- SYNOPSIS ..................................................................................................................................541
- STANDARD OPTIONS ..................................................................................................................541
- WIDGET–SPECIFIC OPTIONS ........................................................................................................541
- INTRODUCTION ..........................................................................................................................543
- COMMAND ENTRY E N T R I E S ..................................................................................................544
- SEPARATOR ENTRY E N T R I E S ..................................................................................................544
- CHECKBUTTON ENTRY E N T R I E S ............................................................................................544
- RADIOBUTTON ENTRY E N T R I E S ............................................................................................544
- CASCADE ENTRY E N T R I E S ......................................................................................................545
- TEAR−OFF ENTRY E N T R I E S ......................................................................................................545
- MENUBARS ...................................................................................................................................545
- SPECIAL MENUS IN MENUBARS ...............................................................................................545
- CLONES .........................................................................................................................................545
- WIDGET COMMAND .....................................................................................................................545
- MENU CONFIGURATIONS ..............................................................................................................549
- DEFAULT BINDINGS .......................................................................................................................551
- BUGS ..............................................................................................................................................551
- KEYWORDS ....................................................................................................................................551

### message

- NAME ...........................................................................................................................................546
- SYNOPSIS ..................................................................................................................................546
- STANDARD OPTIONS ..................................................................................................................547
- WIDGET–SPECIFIC OPTIONS ........................................................................................................547
- DESCRIPTION .............................................................................................................................548
- WIDGET COMMAND .....................................................................................................................548
- DEFAULT BINDINGS .......................................................................................................................549
- BUGS ..............................................................................................................................................549
- KEYWORDS ....................................................................................................................................549

### option

- NAME ...........................................................................................................................................550
- KEYWORDS ....................................................................................................................................550
Table of Contents

Tk Commands

SYNOPSIS ................................................................................................................................. 550
DESCRIPTION ............................................................................................................................ 550
EXAMPLES ................................................................................................................................. 551
KEYWORDS ............................................................................................................................... 551
options ....................................................................................................................................... 553
NAME ......................................................................................................................................... 554
DESCRIPTION .......................................................................................................................... 554
SEE ALSO .................................................................................................................................. 561
KEYWORDS ............................................................................................................................... 561
pack ............................................................................................................................................ 562
NAME ......................................................................................................................................... 562
SYNOPSIS ................................................................................................................................. 563
DESCRIPTION .......................................................................................................................... 563
THE PACKER ALGORITHM ..................................................................................................... 564
EXPANSION ............................................................................................................................. 565
GEOMETRY PROPAGATION ................................................................................................. 566
RESTRICTIONS ON MASTER WINDOWS ............................................................................... 566
PACKING ORDER .................................................................................................................... 566
EXAMPLE .................................................................................................................................. 566
SEE ALSO .................................................................................................................................. 566
KEYWORDS ............................................................................................................................... 567
panedwindow ............................................................................................................................ 568
NAME ......................................................................................................................................... 569
SYNOPSIS ................................................................................................................................. 569
STANDARD OPTIONS ............................................................................................................. 569
WIDGET–SPECIFIC OPTIONS ............................................................................................... 569
DESCRIPTION .......................................................................................................................... 570
WIDGET COMMAND ............................................................................................................... 570
RESIZING PANES ..................................................................................................................... 573
KEYWORDS ............................................................................................................................... 573
photo ........................................................................................................................................... 574
NAME ......................................................................................................................................... 575
SYNOPSIS ................................................................................................................................. 575
DESCRIPTION .......................................................................................................................... 575
CREATING PHOTOS ................................................................................................................ 575
IMAGE COMMAND ................................................................................................................ 576
IMAGE FORMATS .................................................................................................................... 580
COLOR ALLOCATION ............................................................................................................. 580
CREDITS .................................................................................................................................. 581
EXAMPLE .................................................................................................................................. 581
SEE ALSO .................................................................................................................................. 581
KEYWORDS ............................................................................................................................... 581
## Table of Contents

**Tk Commands**

**place**

- NAME .......................................................................................................................... 582
- SYNOPSIS ..................................................................................................................... 582
- DESCRIPTION ................................................................................................................. 582
- FINE POINTS .................................................................................................................. 585
- EXAMPLE ....................................................................................................................... 585
- SEE ALSO ....................................................................................................................... 585
- KEYWORDS .................................................................................................................... 586

**radiobutton**

- NAME .......................................................................................................................... 588
- SYNOPSIS ..................................................................................................................... 588
- STANDARD OPTIONS .................................................................................................... 588
- WIDGET–SPECIFIC OPTIONS ......................................................................................... 589
- DESCRIPTION ................................................................................................................. 590
- WIDGET COMMAND ....................................................................................................... 591
- BINDINGS ....................................................................................................................... 592
- SEE ALSO ....................................................................................................................... 592
- KEYWORDS .................................................................................................................... 592

**raise**

- NAME .......................................................................................................................... 593
- SYNOPSIS ..................................................................................................................... 593
- DESCRIPTION ................................................................................................................. 593
- EXAMPLE ....................................................................................................................... 593
- SEE ALSO ....................................................................................................................... 593
- KEYWORDS .................................................................................................................... 593

**scale**

- NAME .......................................................................................................................... 595
- SYNOPSIS ..................................................................................................................... 595
- STANDARD OPTIONS .................................................................................................... 595
- WIDGET–SPECIFIC OPTIONS ......................................................................................... 595
- DESCRIPTION ................................................................................................................. 597
- WIDGET COMMAND ....................................................................................................... 598
- BINDINGS ....................................................................................................................... 599
- KEYWORDS .................................................................................................................... 599

**scrollbar**

- NAME .......................................................................................................................... 600
- SYNOPSIS ..................................................................................................................... 600
- STANDARD OPTIONS .................................................................................................... 600
- WIDGET–SPECIFIC OPTIONS ......................................................................................... 600
- DESCRIPTION ................................................................................................................. 602
- ELEMENTS ..................................................................................................................... 602
- WIDGET COMMAND ....................................................................................................... 603
# Table of Contents

## Tk Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCROLLING COMMANDS</td>
<td>604</td>
</tr>
<tr>
<td>OLD COMMAND SYNTAX</td>
<td>604</td>
</tr>
<tr>
<td>BINDINGS</td>
<td>605</td>
</tr>
<tr>
<td>EXAMPLE</td>
<td>606</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>606</td>
</tr>
<tr>
<td>selection</td>
<td>608</td>
</tr>
<tr>
<td>NAME</td>
<td>608</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>608</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>608</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>610</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>610</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>611</td>
</tr>
<tr>
<td>send</td>
<td>612</td>
</tr>
<tr>
<td>NAME</td>
<td>612</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>612</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>612</td>
</tr>
<tr>
<td>APPLICATION NAMES</td>
<td>613</td>
</tr>
<tr>
<td>DISABLING SENDS</td>
<td>613</td>
</tr>
<tr>
<td>SECURITY</td>
<td>613</td>
</tr>
<tr>
<td>EXAMPLE</td>
<td>613</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>614</td>
</tr>
<tr>
<td>spinbox</td>
<td>615</td>
</tr>
<tr>
<td>NAME</td>
<td>617</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>617</td>
</tr>
<tr>
<td>STANDARD OPTIONS</td>
<td>618</td>
</tr>
<tr>
<td>WIDGET–SPECIFIC OPTIONS</td>
<td>618</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>620</td>
</tr>
<tr>
<td>VALIDATION</td>
<td>621</td>
</tr>
<tr>
<td>WIDGET COMMAND</td>
<td>622</td>
</tr>
<tr>
<td>DEFAULT BINDINGS</td>
<td>625</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>627</td>
</tr>
<tr>
<td>text</td>
<td>628</td>
</tr>
<tr>
<td>NAME</td>
<td>632</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>632</td>
</tr>
<tr>
<td>STANDARD OPTIONS</td>
<td>632</td>
</tr>
<tr>
<td>WIDGET–SPECIFIC OPTIONS</td>
<td>633</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>634</td>
</tr>
<tr>
<td>INDICES</td>
<td>635</td>
</tr>
<tr>
<td>TAGS</td>
<td>637</td>
</tr>
<tr>
<td>MARKS</td>
<td>639</td>
</tr>
<tr>
<td>EMBEDDED WINDOWS</td>
<td>640</td>
</tr>
<tr>
<td>EMBEDDED IMAGES</td>
<td>641</td>
</tr>
</tbody>
</table>
# Table of Contents

**Tk Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>THE SELECTION</td>
<td>642</td>
</tr>
<tr>
<td>THE INSERTION CURSOR</td>
<td>642</td>
</tr>
<tr>
<td>THE MODIFIED FLAG</td>
<td>642</td>
</tr>
<tr>
<td>THE UNDO MECHANISM</td>
<td>643</td>
</tr>
<tr>
<td>WIDGET COMMAND</td>
<td>643</td>
</tr>
<tr>
<td>BINDINGS</td>
<td>654</td>
</tr>
<tr>
<td>PERFORMANCE ISSUES</td>
<td>656</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>657</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>658</td>
</tr>
<tr>
<td>tk</td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td>658</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>658</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>658</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>659</td>
</tr>
<tr>
<td>chooseColor</td>
<td>660</td>
</tr>
<tr>
<td>NAME</td>
<td>660</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>660</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>660</td>
</tr>
<tr>
<td>EXAMPLE</td>
<td>660</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>660</td>
</tr>
<tr>
<td>chooseDirectory</td>
<td>661</td>
</tr>
<tr>
<td>NAME</td>
<td>661</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>661</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>661</td>
</tr>
<tr>
<td>EXAMPLE</td>
<td>662</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>662</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>662</td>
</tr>
<tr>
<td>dialog</td>
<td>663</td>
</tr>
<tr>
<td>NAME</td>
<td>663</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>663</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>663</td>
</tr>
<tr>
<td>EXAMPLE</td>
<td>664</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>664</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>664</td>
</tr>
<tr>
<td>focusNext</td>
<td>665</td>
</tr>
<tr>
<td>NAME</td>
<td>665</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>665</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>665</td>
</tr>
<tr>
<td>getOpenFile</td>
<td>666</td>
</tr>
<tr>
<td>NAME</td>
<td>666</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>666</td>
</tr>
</tbody>
</table>
Table of Contents

Tk Commands

DESCRIPTION ................................................................................................................................. 666
SPECIFYING FILE PATTERNS ........................................................................................................ 667
SPECIFYING EXTENSIONS ........................................................................................................... 668
EXAMPLE ..................................................................................................................................... 668
SEE ALSO ..................................................................................................................................... 669
KEYWORDS ................................................................................................................................. 669
messageBox ................................................................................................................................... 670
 NAME ........................................................................................................................................ 670
 SYNOPSIS .................................................................................................................................. 670
 DESCRIPTION ............................................................................................................................ 670
 EXAMPLE ..................................................................................................................................... 671
 KEYWORDS .................................................................................................................................. 671
optionMenu ..................................................................................................................................... 672
 NAME ........................................................................................................................................ 672
 SYNOPSIS .................................................................................................................................. 672
 DESCRIPTION ............................................................................................................................ 672
 KEYWORDS .................................................................................................................................. 672
popup ............................................................................................................................................... 673
 NAME ........................................................................................................................................ 673
 SYNOPSIS .................................................................................................................................. 673
 DESCRIPTION ............................................................................................................................ 673
 EXAMPLE ..................................................................................................................................... 673
 SEE ALSO .................................................................................................................................... 673
 KEYWORDS .................................................................................................................................. 673
palette ............................................................................................................................................. 674
 NAME ........................................................................................................................................ 674
 SYNOPSIS .................................................................................................................................. 674
 DESCRIPTION ............................................................................................................................ 674
 KEYWORDS .................................................................................................................................. 675
tkerror ............................................................................................................................................ 676
 NAME ........................................................................................................................................ 676
 SYNOPSIS .................................................................................................................................. 676
 DESCRIPTION ............................................................................................................................ 676
 KEYWORDS .................................................................................................................................. 676
tkvars ............................................................................................................................................... 677
 NAME ........................................................................................................................................ 677
 DESCRIPTION ............................................................................................................................ 677
 KEYWORDS .................................................................................................................................. 678
tkwait ............................................................................................................................................... 679
 NAME ........................................................................................................................................ 679
 SYNOPSIS .................................................................................................................................. 679
 DESCRIPTION ............................................................................................................................ 679
# Table of Contents

## Tk Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>toplevel</td>
<td>DESCRIPTION</td>
<td>680</td>
</tr>
<tr>
<td></td>
<td>NAME</td>
<td>680</td>
</tr>
<tr>
<td></td>
<td>SYNOPSIS</td>
<td>681</td>
</tr>
<tr>
<td></td>
<td>STANDARD OPTIONS</td>
<td>681</td>
</tr>
<tr>
<td></td>
<td>WIDGET–SPECIFIC OPTIONS</td>
<td>681</td>
</tr>
<tr>
<td></td>
<td>DESCRIPTION</td>
<td>681</td>
</tr>
<tr>
<td></td>
<td>WIDGET COMMAND</td>
<td>683</td>
</tr>
<tr>
<td></td>
<td>BINDINGS</td>
<td>683</td>
</tr>
<tr>
<td></td>
<td>SEE ALSO</td>
<td>683</td>
</tr>
<tr>
<td></td>
<td>KEYWORDS</td>
<td>684</td>
</tr>
<tr>
<td>wininfo</td>
<td>DESCRIPTION</td>
<td>686</td>
</tr>
<tr>
<td></td>
<td>NAME</td>
<td>686</td>
</tr>
<tr>
<td></td>
<td>SYNOPSIS</td>
<td>686</td>
</tr>
<tr>
<td></td>
<td>DESCRIPTION</td>
<td>686</td>
</tr>
<tr>
<td></td>
<td>EXAMPLE</td>
<td>690</td>
</tr>
<tr>
<td></td>
<td>KEYWORDS</td>
<td>689</td>
</tr>
<tr>
<td>wm</td>
<td>DESCRIPTION</td>
<td>691</td>
</tr>
<tr>
<td></td>
<td>NAME</td>
<td>692</td>
</tr>
<tr>
<td></td>
<td>SYNOPSIS</td>
<td>692</td>
</tr>
<tr>
<td></td>
<td>DESCRIPTION</td>
<td>692</td>
</tr>
<tr>
<td></td>
<td>GEOMETRY MANAGEMENT</td>
<td>698</td>
</tr>
<tr>
<td></td>
<td>GRIDDED GEOMETRY MANAGEMENT</td>
<td>698</td>
</tr>
<tr>
<td></td>
<td>BUGS</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td>EXAMPLES</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td>SEE ALSO</td>
<td>700</td>
</tr>
<tr>
<td></td>
<td>KEYWORDS</td>
<td>700</td>
</tr>
</tbody>
</table>

## Tcl Library

<table>
<thead>
<tr>
<th>Command</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileSystem</td>
<td>DESCRIPTION</td>
<td>705</td>
</tr>
<tr>
<td></td>
<td>NAME</td>
<td>706</td>
</tr>
<tr>
<td></td>
<td>SYNOPSIS</td>
<td>706</td>
</tr>
<tr>
<td></td>
<td>ARGUMENTS</td>
<td>708</td>
</tr>
<tr>
<td></td>
<td>DESCRIPTION</td>
<td>710</td>
</tr>
<tr>
<td></td>
<td>TCL_FILESYSTEM</td>
<td>716</td>
</tr>
<tr>
<td></td>
<td>TYPENAME</td>
<td>718</td>
</tr>
<tr>
<td></td>
<td>STRUCTURE LENGTH</td>
<td>718</td>
</tr>
<tr>
<td></td>
<td>VERSION</td>
<td>719</td>
</tr>
<tr>
<td></td>
<td>FILESYSTEM INFRASTRUCTURE</td>
<td>719</td>
</tr>
<tr>
<td></td>
<td>PATHINFOFILESYSTEMPROC</td>
<td>719</td>
</tr>
<tr>
<td></td>
<td>DUPINTERNALREPPROC</td>
<td>719</td>
</tr>
<tr>
<td></td>
<td>FREEINTERNALREPPROC</td>
<td>719</td>
</tr>
</tbody>
</table>
# Table of Contents

## Tcl Library

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERNALTONORMALIZEDPROC</td>
<td>719</td>
</tr>
<tr>
<td>CREATEINTERNALREPPROC</td>
<td>720</td>
</tr>
<tr>
<td>NORMALIZEPATHPROC</td>
<td>720</td>
</tr>
<tr>
<td>FILESYSTEM OPERATIONS</td>
<td>720</td>
</tr>
<tr>
<td>FILESYSTEMPATHTYPEPROC</td>
<td>721</td>
</tr>
<tr>
<td>FILESYSTEMSEPARATORPROC</td>
<td>721</td>
</tr>
<tr>
<td>STATPROC</td>
<td>721</td>
</tr>
<tr>
<td>ACCESSPROC</td>
<td>721</td>
</tr>
<tr>
<td>OPENFILECHANNELPROC</td>
<td>722</td>
</tr>
<tr>
<td>MATCHINDIRECTORYPROC</td>
<td>722</td>
</tr>
<tr>
<td>UTIMEPROC</td>
<td>723</td>
</tr>
<tr>
<td>LINKPROC</td>
<td>723</td>
</tr>
<tr>
<td>LISTVOLUMESPROC</td>
<td>724</td>
</tr>
<tr>
<td>FILEATTRSTRINGSPROC</td>
<td>724</td>
</tr>
<tr>
<td>FILEATTRSGETPROC</td>
<td>724</td>
</tr>
<tr>
<td>FILEATTRSSETPROC</td>
<td>725</td>
</tr>
<tr>
<td>CREATEDIRECTORYPROC</td>
<td>725</td>
</tr>
<tr>
<td>REMOVEDIRECTORYPROC</td>
<td>725</td>
</tr>
<tr>
<td>DELETEFILEPROC</td>
<td>725</td>
</tr>
<tr>
<td>FILESYSTEM EFFICIENCY</td>
<td>726</td>
</tr>
<tr>
<td>LSTATPROC</td>
<td>726</td>
</tr>
<tr>
<td>COPYFILEPROC</td>
<td>726</td>
</tr>
<tr>
<td>RENAMEFILEPROC</td>
<td>726</td>
</tr>
<tr>
<td>COPYDIRECTORYPROC</td>
<td>727</td>
</tr>
<tr>
<td>LOADFILEPROC</td>
<td>727</td>
</tr>
<tr>
<td>UNLOADFILEPROC</td>
<td>727</td>
</tr>
<tr>
<td>GETCWDPROC</td>
<td>728</td>
</tr>
<tr>
<td>CHDIRPROC</td>
<td>728</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>728</td>
</tr>
</tbody>
</table>

## Notifier

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>729</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>729</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>730</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>731</td>
</tr>
<tr>
<td>NOTIFIER BASICS</td>
<td>732</td>
</tr>
<tr>
<td>CREATING A NEW EVENT SOURCE</td>
<td>732</td>
</tr>
<tr>
<td>CREATING A NEW NOTIFIER</td>
<td>736</td>
</tr>
<tr>
<td>EXTERNAL EVENT LOOPS</td>
<td>737</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>738</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>738</td>
</tr>
</tbody>
</table>

## StdChannels

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>739</td>
</tr>
</tbody>
</table>

---

xxxii
Table of Contents

Tcl Library

DESCRIPTION ................................................................................................................................................. 739
APIs .......................................................................................................................................................... 739
INITIALIZATION OF TCL STANDARD CHANNELS ............................................................................... 740
RE-INITIALIZATION OF TCL STANDARD CHANNELS ........................................................................ 740
tclsh .......................................................................................................................................................... 741
wish ............................................................................................................................................................ 741
SEE ALSO .................................................................................................................................................. 741
KEYWORDS ............................................................................................................................................... 741

TCL_MEM_DEBUG........................................................................................................................................ 742
NAME ........................................................................................................................................................ 742
DESCRIPTION .......................................................................................................................................... 742
ENABLING MEMORY DEBUGGING ........................................................................................................ 742
GUARD ZONES ......................................................................................................................................... 742
DEBUGGING DIFFICULT MEMORY CORRUPTION PROBLEMS .......................................................... 742
SEE ALSO .................................................................................................................................................. 743
KEYWORDS ............................................................................................................................................... 743

Access .......................................................................................................................................................... 744
NAME ........................................................................................................................................................ 744
SYNOPSIS ............................................................................................................................................... 744
ARGUMENTS .......................................................................................................................................... 744
DESCRIPTION .......................................................................................................................................... 744
KEYWORDS ............................................................................................................................................... 745

AddErrInfo .................................................................................................................................................. 746
NAME ........................................................................................................................................................ 746
SYNOPSIS ............................................................................................................................................... 746
ARGUMENTS .......................................................................................................................................... 746
DESCRIPTION .......................................................................................................................................... 747
SEE ALSO .................................................................................................................................................. 748
KEYWORDS ............................................................................................................................................... 748

Alloc ........................................................................................................................................................... 749
NAME ........................................................................................................................................................ 749
SYNOPSIS ............................................................................................................................................... 749
ARGUMENTS .......................................................................................................................................... 749
DESCRIPTION .......................................................................................................................................... 749
KEYWORDS ............................................................................................................................................... 750

AllowExc ..................................................................................................................................................... 751
NAME ........................................................................................................................................................ 751
SYNOPSIS ............................................................................................................................................... 751
ARGUMENTS .......................................................................................................................................... 751
DESCRIPTION .......................................................................................................................................... 751
KEYWORDS ............................................................................................................................................... 751

Applnit .......................................................................................................................................................... 752
# Table of Contents

## Tcl Library

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>752</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>752</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>752</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>752</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>753</td>
</tr>
</tbody>
</table>

**Async**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>754</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>754</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>754</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>754</td>
</tr>
<tr>
<td>WARNING</td>
<td>756</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>756</td>
</tr>
</tbody>
</table>

**BackedErr**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>757</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>757</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>757</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>757</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>757</td>
</tr>
</tbody>
</table>

**Backslash**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>758</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>758</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>758</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>758</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>758</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>758</td>
</tr>
</tbody>
</table>

**BoolObj**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>759</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>759</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>759</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>759</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>760</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>760</td>
</tr>
</tbody>
</table>

**ByteArrObj**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>761</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>761</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>761</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>761</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>762</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>762</td>
</tr>
</tbody>
</table>

**CallDel**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>763</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>763</td>
</tr>
</tbody>
</table>
# Table of Contents

**Tcl Library**

<table>
<thead>
<tr>
<th>Function</th>
<th>Arguments</th>
<th>Description</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>CmdCmplt</td>
<td></td>
<td></td>
<td>765</td>
</tr>
<tr>
<td>Concat</td>
<td></td>
<td></td>
<td>766</td>
</tr>
<tr>
<td>CrtChannel</td>
<td></td>
<td></td>
<td>768</td>
</tr>
<tr>
<td>CrtChnlHdlr</td>
<td></td>
<td></td>
<td>783</td>
</tr>
</tbody>
</table>

**SYNOPSIS**

**NAME**

**KEYWORDS**

**SEE ALSO**

**OLD CHANNEL TYPES**

- TCL_BADCHANNELOPTION
- HANDLERPROC
- FLUSHPROC
- WATCHPROC
- GEHANDLEPROC
- SECOPTIONPROC
- GETOPTIONPROC
- TCL_BADCHANNELOPTION
- OLD CHANNEL TYPES
- SEE ALSO
- KEYWORDS
Table of Contents

Tcl Library

ARGUMENTS ........................................................................................................................................... 783
DESCRIPTION ........................................................................................................................................ 783
SEE ALSO .......................................................................................................................................... 784
KEYWORDS ......................................................................................................................................... 784
CrtCloseHdlr ........................................................................................................................................ 785
NAME ............................................................................................................................................... 785
SYNOPSIS ....................................................................................................................................... 785
ARGUMENTS ................................................................................................................................... 785
DESCRIPTION ............................................................................................................................... 785
SEE ALSO ...................................................................................................................................... 785
KEYWORDS ................................................................................................................................... 786
CrtCommand .................................................................................................................................... 787
NAME ............................................................................................................................................ 787
SYNOPSIS ...................................................................................................................................... 787
ARGUMENTS ................................................................................................................................... 787
DESCRIPTION ............................................................................................................................... 787
SEE ALSO ...................................................................................................................................... 789
KEYWORDS ................................................................................................................................... 789
CrtFileHdlr ....................................................................................................................................... 790
NAME ............................................................................................................................................ 790
SYNOPSIS ...................................................................................................................................... 790
ARGUMENTS ................................................................................................................................... 790
DESCRIPTION ............................................................................................................................... 790
KEYWORDS ................................................................................................................................... 791
CrtInterp ........................................................................................................................................ 792
NAME ............................................................................................................................................ 792
SYNOPSIS ...................................................................................................................................... 792
ARGUMENTS ................................................................................................................................... 792
DESCRIPTION ............................................................................................................................... 792
INTERPRETERS AND MEMORY MANAGEMENT ............................................................................ 793
SEE ALSO ...................................................................................................................................... 794
KEYWORDS ................................................................................................................................... 794
CrtMathFcns .................................................................................................................................... 795
NAME ............................................................................................................................................ 795
SYNOPSIS ...................................................................................................................................... 795
ARGUMENTS ................................................................................................................................... 795
DESCRIPTION ............................................................................................................................... 796
KEYWORDS ................................................................................................................................... 797
SEE ALSO ...................................................................................................................................... 797
CrtObjCmnd ..................................................................................................................................... 798
NAME ............................................................................................................................................ 798
SYNOPSIS ...................................................................................................................................... 798
# Table of Contents

**Tcl Library**

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARGUMENTS</td>
<td></td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td></td>
</tr>
<tr>
<td>SEE ALSO</td>
<td></td>
</tr>
<tr>
<td>KEYWORDS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CrtSlave</td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td></td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td></td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td></td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td></td>
</tr>
<tr>
<td>SEE ALSO</td>
<td></td>
</tr>
<tr>
<td>KEYWORDS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CrtTimerHdr</td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td></td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td></td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td></td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td></td>
</tr>
<tr>
<td>KEYWORDS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CrtTrace</td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td></td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td></td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td></td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td></td>
</tr>
<tr>
<td>KEYWORDS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DString</td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td></td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td></td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td></td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td></td>
</tr>
<tr>
<td>KEYWORDS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DetachPids</td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td></td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td></td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td></td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td></td>
</tr>
<tr>
<td>KEYWORDS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoOneEvent</td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td></td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td></td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td></td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td></td>
</tr>
<tr>
<td>KEYWORDS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoWhenIdle</td>
<td></td>
</tr>
</tbody>
</table>

---

xxxvii
## Table of Contents

**Tcl Library**

<table>
<thead>
<tr>
<th>NAME</th>
<th>SYNOPSIS</th>
<th>ARGUMENTS</th>
<th>DESCRIPTION</th>
<th>BUGS</th>
<th>KEYWORDS</th>
<th>SEE ALSO</th>
<th>KEYWORDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eval</td>
<td>825</td>
<td>825</td>
<td>825</td>
<td>826</td>
<td>826</td>
<td>826</td>
<td>826</td>
</tr>
<tr>
<td>Exit</td>
<td>829</td>
<td>829</td>
<td>829</td>
<td>829</td>
<td>829</td>
<td>829</td>
<td>829</td>
</tr>
<tr>
<td>ExprLong</td>
<td>831</td>
<td>831</td>
<td>831</td>
<td>831</td>
<td>831</td>
<td>831</td>
<td>831</td>
</tr>
<tr>
<td>ExprLongObj</td>
<td>833</td>
<td>833</td>
<td>833</td>
<td>833</td>
<td>833</td>
<td>833</td>
<td>833</td>
</tr>
</tbody>
</table>

**DoubleObj**

<table>
<thead>
<tr>
<th>NAME</th>
<th>SYNOPSIS</th>
<th>ARGUMENTS</th>
<th>DESCRIPTION</th>
<th>SEE ALSO</th>
<th>KEYWORDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>821</td>
<td>821</td>
<td>821</td>
<td>821</td>
<td>822</td>
<td>822</td>
</tr>
</tbody>
</table>

**DumpActiveMemory**

<table>
<thead>
<tr>
<th>NAME</th>
<th>SYNOPSIS</th>
<th>ARGUMENTS</th>
<th>DESCRIPTION</th>
<th>SEE ALSO</th>
<th>KEYWORDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>823</td>
<td>823</td>
<td>823</td>
<td>823</td>
<td>824</td>
<td>824</td>
</tr>
</tbody>
</table>

**ExprLong**

<table>
<thead>
<tr>
<th>NAME</th>
<th>SYNOPSIS</th>
<th>ARGUMENTS</th>
<th>DESCRIPTION</th>
<th>SEE ALSO</th>
<th>KEYWORDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>831</td>
<td>831</td>
<td>831</td>
<td>831</td>
<td>832</td>
<td>832</td>
</tr>
</tbody>
</table>

**ExprLongObj**

<table>
<thead>
<tr>
<th>NAME</th>
<th>SYNOPSIS</th>
<th>ARGUMENTS</th>
<th>DESCRIPTION</th>
<th>SEE ALSO</th>
<th>KEYWORDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>833</td>
<td>833</td>
<td>833</td>
<td>833</td>
<td>833</td>
<td>833</td>
</tr>
</tbody>
</table>
# Table of Contents

## Tcl Library

- **NAME** ........................................................................................................................................... 833
- **SYNOPSIS** .................................................................................................................................... 833
- **ARGUMENTS** ............................................................................................................................. 833
- **DESCRIPTION** ............................................................................................................................ 833
- **SEE ALSO** .................................................................................................................................... 834
- **KEYWORDS** .................................................................................................................................... 834

## FindExec

- **NAME** ........................................................................................................................................... 835
- **SYNOPSIS** .................................................................................................................................... 835
- **ARGUMENTS** ............................................................................................................................. 835
- **DESCRIPTION** ............................................................................................................................ 835
- **KEYWORDS** .................................................................................................................................... 835

## GetCwd

- **NAME** ........................................................................................................................................... 836
- **SYNOPSIS** .................................................................................................................................... 836
- **ARGUMENTS** ............................................................................................................................. 836
- **DESCRIPTION** ............................................................................................................................ 836
- **KEYWORDS** .................................................................................................................................... 836

## Encoding

- **NAME** ........................................................................................................................................... 837
- **SYNOPSIS** .................................................................................................................................... 837
- **ARGUMENTS** ............................................................................................................................. 838
- **INTRODUCTION** .......................................................................................................................... 839
- **DESCRIPTION** ............................................................................................................................ 840
- **ENCODING FILES** ....................................................................................................................... 843
- **KEYWORDS** .................................................................................................................................... 846

## GetHostName

- **NAME** ........................................................................................................................................... 847
- **SYNOPSIS** .................................................................................................................................... 847
- **DESCRIPTION** ............................................................................................................................ 847

## GetIndex

- **NAME** ........................................................................................................................................... 848
- **SYNOPSIS** .................................................................................................................................... 848
- **DESCRIPTION** ............................................................................................................................. 848
- **SEE ALSO** ..................................................................................................................................... 849
- **KEYWORDS** ..................................................................................................................................... 849

## GetInt

- **NAME** ........................................................................................................................................... 850
- **SYNOPSIS** .................................................................................................................................... 850
- **ARGUMENTS** ............................................................................................................................. 850
# Table of Contents

## Tcl Library

**DESCRIPTION** .......................................................................................................................................................... 850

**KEYWORDS** .......................................................................................................................................................... 851

**GetOpnFl** .............................................................................................................................................................. 852

**NAME** ................................................................................................................................................................. 852

**SYNOPSIS** ............................................................................................................................................................ 852

**ARGUMENTS** ........................................................................................................................................................ 852

**DESCRIPTION** ...................................................................................................................................................... 852

**SEE ALSO** ........................................................................................................................................................... 853

**KEYWORDS** .......................................................................................................................................................... 852

**GetStdChan** ........................................................................................................................................................ 853

**NAME** ................................................................................................................................................................. 853

**SYNOPSIS** ............................................................................................................................................................ 853

**ARGUMENTS** ........................................................................................................................................................ 853

**DESCRIPTION** ...................................................................................................................................................... 853

**SEE ALSO** ........................................................................................................................................................... 854

**KEYWORDS** .......................................................................................................................................................... 854

**GetTime** ............................................................................................................................................................... 855

**NAME** ................................................................................................................................................................. 855

**SYNOPSIS** ............................................................................................................................................................ 855

**ARGUMENTS** ........................................................................................................................................................ 855

**DESCRIPTION** ...................................................................................................................................................... 855

**SEE ALSO** ........................................................................................................................................................... 855

**KEYWORDS** .......................................................................................................................................................... 855

**GetVersion** ........................................................................................................................................................ 857

**NAME** ................................................................................................................................................................. 857

**SYNOPSIS** ............................................................................................................................................................ 857

**ARGUMENTS** ........................................................................................................................................................ 857

**DESCRIPTION** ...................................................................................................................................................... 857

**KEYWORDS** .......................................................................................................................................................... 857

**Hash** ..................................................................................................................................................................... 858

**NAME** ................................................................................................................................................................. 858

**SYNOPSIS** ............................................................................................................................................................ 858

**ARGUMENTS** ........................................................................................................................................................ 859

**DESCRIPTION** ...................................................................................................................................................... 859

**THE TCL_HASHKEYTYPE STRUCTURE** ........................................................................................................... 861

**KEYWORDS** .......................................................................................................................................................... 863

**Init** ....................................................................................................................................................................... 864

**NAME** ................................................................................................................................................................. 864

**SYNOPSIS** ............................................................................................................................................................ 864

**ARGUMENTS** ........................................................................................................................................................ 864

**DESCRIPTION** ...................................................................................................................................................... 864

**SEE ALSO** ........................................................................................................................................................... 864

**KEYWORDS** .......................................................................................................................................................... 864
Table of Contents

Tcl Library

InitStubs.................................................................................................................................865
  NAME.................................................................................................................................865
  SYNOPSIS..........................................................................................................................865
  ARGUMENTS......................................................................................................................865
  INTRODUCTION...............................................................................................................865
  DESCRIPTION....................................................................................................................866
  SEE ALSO..........................................................................................................................866
  KEYWORDS.........................................................................................................................867

IntObj........................................................................................................................................868
  NAME.................................................................................................................................868
  SYNOPSIS..........................................................................................................................868
  ARGUMENTS......................................................................................................................868
  DESCRIPTION....................................................................................................................869
  SEE ALSO..........................................................................................................................869
  KEYWORDS.........................................................................................................................869

Interp.......................................................................................................................................870
  NAME.................................................................................................................................870
  SYNOPSIS..........................................................................................................................870
  DESCRIPTION....................................................................................................................870
  KEYWORDS.........................................................................................................................871

LinkVar..................................................................................................................................872
  NAME.................................................................................................................................872
  SYNOPSIS..........................................................................................................................872
  ARGUMENTS......................................................................................................................872
  DESCRIPTION....................................................................................................................873
  KEYWORDS.........................................................................................................................874

ListObj....................................................................................................................................875
  NAME.................................................................................................................................875
  SYNOPSIS..........................................................................................................................875
  ARGUMENTS......................................................................................................................875
  DESCRIPTION....................................................................................................................876
  SEE ALSO..........................................................................................................................878
  KEYWORDS.........................................................................................................................878

Macintosh.............................................................................................................................879
  NAME.................................................................................................................................879
  SYNOPSIS..........................................................................................................................879
  ARGUMENTS......................................................................................................................879
  INTRODUCTION...............................................................................................................880
  DESCRIPTION....................................................................................................................880
  RESOURCE TYPES..........................................................................................................880
  KEYWORDS.........................................................................................................................880

 Tcl_Main.............................................................................................................................882
# Table of Contents

## Tcl Library

<table>
<thead>
<tr>
<th>NAME</th>
<th>SYNOPSIS</th>
<th>ARGUMENTS</th>
<th>DESCRIPTION</th>
<th>SEE ALSO</th>
<th>KEYWORDS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Object

<table>
<thead>
<tr>
<th>NAME</th>
<th>SYNOPSIS</th>
<th>ARGUMENTS</th>
<th>INTRODUCTION</th>
<th>THE TCL_OBJTYPE STRUCTURE</th>
<th>EXAMPLE OF THE LIFETIME OF AN OBJECT</th>
<th>STORAGE MANAGEMENT OF OBJECTS</th>
<th>SEE ALSO</th>
<th>KEYWORDS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## ObjectType

<table>
<thead>
<tr>
<th>NAME</th>
<th>SYNOPSIS</th>
<th>ARGUMENTS</th>
<th>DESCRIPTION</th>
<th>THE TCL_OBJTYPE STRUCTURE</th>
<th>SEE ALSO</th>
<th>KEYWORDS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## OpenFileChnl

<table>
<thead>
<tr>
<th>NAME</th>
<th>SYNOPSIS</th>
<th>ARGUMENTS</th>
<th>DESCRIPTION</th>
<th>TCL_OPENFILECHANNEL</th>
<th>TCL OPENCOMMANDCHANNEL</th>
<th>TCL_MAKEFILECHANNEL</th>
<th>TCL_GETCHANNEL</th>
<th>TCL_REGISTERCHANNEL</th>
<th>TCL_UNREGISTERCHANNEL</th>
<th>TCL_DETACHCHANNEL</th>
<th>TCL ISSTANDARDCHANNEL</th>
<th>TCL CLOSE</th>
<th>TCL READCHARS AND TCL_READ</th>
<th>TCL GETSOBJ AND TCL GETS</th>
<th>TCL UNGETS</th>
<th>TCL WRITECHARS, TCL_WRITEOBJ, AND TCL_WRITE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table of Contents

Tcl Library

TCL_FLUSH ................................................................. 903
TCL_SEEK ................................................................. 903
TCL_TELL ................................................................. 903
TCL_GETCHANNELOPTION ........................................ 904
TCL_SETCHANNELOPTION ........................................ 904
TCL_EOF ................................................................. 904
TCL_INPUTBLOCKED .................................................. 904
TCL_INPUTBUFFERED ................................................ 904
TCL_OUTPUTBUFFERED .............................................. 904
PLATFORM ISSUES ................................................... 904
SEE ALSO ........................................................................ 905
KEYWORDS ..................................................................... 905

OpenTcp ................................................................. 906

NAME ........................................................................... 906
SYNOPSIS ..................................................................... 906
ARGUMENTS .................................................................. 906
DESCRIPTION .......................................................... 907
TCL_OPENTCPClient ................................................... 907
TCL_MAKETCPCLIENTCHANNEL .................................. 908
TCL_OPENTCP SERVER ............................................... 908
PLATFORM ISSUES ................................................... 909
SEE ALSO ........................................................................ 909
KEYWORDS ..................................................................... 909

Panic ................................................................. 910

NAME ........................................................................... 910
SYNOPSIS ..................................................................... 910
ARGUMENTS .................................................................. 910
DESCRIPTION .......................................................... 910
SEE ALSO ........................................................................ 911
KEYWORDS ..................................................................... 911

ParseCmd ................................................................. 912

NAME ........................................................................... 912
SYNOPSIS ..................................................................... 912
ARGUMENTS .................................................................. 913
DESCRIPTION .......................................................... 913
TCL_PARSE STRUCTURE ............................................ 915
KEYWORDS ..................................................................... 918

PkgRequire ................................................................. 919

NAME ........................................................................... 919
SYNOPSIS ..................................................................... 919
ARGUMENTS .................................................................. 919
DESCRIPTION .......................................................... 920
# Table of Contents

## Tcl Library

<table>
<thead>
<tr>
<th>Key</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYWORDS</td>
<td>920</td>
</tr>
<tr>
<td>Preserve</td>
<td>921</td>
</tr>
<tr>
<td>NAME</td>
<td>921</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>921</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>921</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>921</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>922</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>922</td>
</tr>
</tbody>
</table>

- **PrintDb**: 923
  - NAME       | 923  |
  - SYNOPSIS   | 923  |
  - ARGUMENTS  | 923  |
  - DESCRIPTION| 923  |
  - KEYWORDS   | 923  |

- **Environment**: 924
  - NAME       | 924  |
  - SYNOPSIS   | 924  |
  - ARGUMENTS  | 924  |
  - DESCRIPTION| 924  |
  - KEYWORDS   | 924  |

- **RecordEval**: 925
  - NAME       | 925  |
  - SYNOPSIS   | 925  |
  - ARGUMENTS  | 925  |
  - DESCRIPTION| 925  |
  - SEE ALSO   | 925  |
  - KEYWORDS   | 926  |

- **RecEvalObj**: 927
  - NAME       | 927  |
  - SYNOPSIS   | 927  |
  - ARGUMENTS  | 927  |
  - DESCRIPTION| 927  |
  - SEE ALSO   | 927  |
  - KEYWORDS   | 927  |

- **RegExp**: 928
  - NAME       | 928  |
  - SYNOPSIS   | 928  |
  - ARGUMENTS  | 929  |
  - DESCRIPTION| 930  |
  - SEE ALSO   | 933  |
  - KEYWORDS   | 933  |

- **SaveResult**: 934
# Table of Contents

## Tcl Library

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>934</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>934</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>934</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>934</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>935</td>
</tr>
<tr>
<td><strong>AssocData</strong></td>
<td>936</td>
</tr>
<tr>
<td>NAME</td>
<td>936</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>936</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>936</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>936</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>937</td>
</tr>
<tr>
<td><strong>SetErrno</strong></td>
<td>938</td>
</tr>
<tr>
<td>NAME</td>
<td>938</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>938</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>938</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>938</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>939</td>
</tr>
<tr>
<td><strong>SetRecLmt</strong></td>
<td>940</td>
</tr>
<tr>
<td>NAME</td>
<td>940</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>940</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>940</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>940</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>940</td>
</tr>
<tr>
<td><strong>SetResult</strong></td>
<td>941</td>
</tr>
<tr>
<td>NAME</td>
<td>941</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>941</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>941</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>941</td>
</tr>
<tr>
<td>OLD STRING PROCEDURES</td>
<td>942</td>
</tr>
<tr>
<td>DIRECT ACCESS TO INTERP−&gt;RESULT IS DEPRECATED</td>
<td>943</td>
</tr>
<tr>
<td><strong>THE TCL_FREEPROC ARGUMENT TO TCL_SETRESULT</strong></td>
<td>943</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>944</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>944</td>
</tr>
<tr>
<td><strong>SetVar</strong></td>
<td>945</td>
</tr>
<tr>
<td>NAME</td>
<td>945</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>945</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>946</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>946</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>948</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>948</td>
</tr>
<tr>
<td><strong>Signal</strong></td>
<td>949</td>
</tr>
<tr>
<td>NAME</td>
<td>949</td>
</tr>
</tbody>
</table>
# Table of Contents

**Tcl Library**

<table>
<thead>
<tr>
<th>NAME</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td>949</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>949</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>949</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>949</td>
</tr>
<tr>
<td>Sleep</td>
<td>950</td>
</tr>
<tr>
<td>NAME</td>
<td>950</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>950</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>950</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>950</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>950</td>
</tr>
<tr>
<td>SourceRCFile</td>
<td>951</td>
</tr>
<tr>
<td>NAME</td>
<td>951</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>951</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>951</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>951</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>951</td>
</tr>
<tr>
<td>SplitList</td>
<td>952</td>
</tr>
<tr>
<td>NAME</td>
<td>952</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>952</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>952</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>953</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>954</td>
</tr>
<tr>
<td>SplitPath</td>
<td>955</td>
</tr>
<tr>
<td>NAME</td>
<td>955</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>955</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>955</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>955</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>956</td>
</tr>
<tr>
<td>ChnlStack</td>
<td>957</td>
</tr>
<tr>
<td>NAME</td>
<td>957</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>957</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>957</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>957</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>958</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>958</td>
</tr>
<tr>
<td>StaticPkg</td>
<td>959</td>
</tr>
<tr>
<td>NAME</td>
<td>959</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>959</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>959</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>959</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>960</td>
</tr>
<tr>
<td>StrMatch</td>
<td>961</td>
</tr>
</tbody>
</table>
Table of Contents

Tcl Library

NAME ..............................................................................................................................................961
SYNOPSIS ...................................................................................................................................961
ARGUMENTS .................................................................................................................................961
DESCRIPTION ...............................................................................................................................961
KEYWORDS .................................................................................................................................961

StringObj ........................................................................................................................................962
NAME .........................................................................................................................................962
SYNOPSIS .................................................................................................................................962
ARGUMENTS ...............................................................................................................................963
DESCRIPTION ............................................................................................................................964
SEE ALSO ....................................................................................................................................966
KEYWORDS ...............................................................................................................................966

SubstObj .........................................................................................................................................967
NAME .........................................................................................................................................967
SYNOPSIS .................................................................................................................................967
ARGUMENTS ...............................................................................................................................967
DESCRIPTION ............................................................................................................................967
SEE ALSO ....................................................................................................................................968
KEYWORDS ...............................................................................................................................968

TraceCmd .......................................................................................................................................969
NAME .........................................................................................................................................969
SYNOPSIS .................................................................................................................................969
ARGUMENTS ...............................................................................................................................969
DESCRIPTION ............................................................................................................................970
CALLING COMMANDS DURING TRACES ...............................................................................971
MULTIPLE TRACES .........................................................................................................................971
TCL_TRACE_DESTROYED FLAG ..................................................................................................971
TCL_INTERP_DESTROYED ..............................................................................................................971
BUGS .......................................................................................................................................972
KEYWORDS ...............................................................................................................................972

TraceVar .......................................................................................................................................973
NAME .........................................................................................................................................973
SYNOPSIS .................................................................................................................................973
ARGUMENTS ...............................................................................................................................974
DESCRIPTION ............................................................................................................................974
TWO–PART NAMES .......................................................................................................................976
ACCESSING VARIABLES DURING TRACES ...............................................................................976
CALLBACK TIMING .........................................................................................................................977
WHOLE–ARRAY TRACES ..............................................................................................................977
MULTIPLE TRACES .........................................................................................................................977
ERROR RETURNS ........................................................................................................................978
RESTRICTIONS ...........................................................................................................................978
# Table of Contents

**Tcl Library**

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNDEFINED VARIABLES</td>
<td>978</td>
</tr>
<tr>
<td>TCL_TRACE_DESTROYED FLAG</td>
<td>978</td>
</tr>
<tr>
<td>TCL_INTERP_DESTROYED</td>
<td>978</td>
</tr>
<tr>
<td>BUGS</td>
<td>979</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>979</td>
</tr>
<tr>
<td>Translate</td>
<td>980</td>
</tr>
<tr>
<td>NAME</td>
<td>980</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>980</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>980</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>980</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>980</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>981</td>
</tr>
<tr>
<td>UniCharIsAlpha</td>
<td>982</td>
</tr>
<tr>
<td>NAME</td>
<td>982</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>982</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>982</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>982</td>
</tr>
<tr>
<td>CHARACTER CLASSES</td>
<td>983</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>983</td>
</tr>
<tr>
<td>UpVar</td>
<td>984</td>
</tr>
<tr>
<td>NAME</td>
<td>984</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>984</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>984</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>984</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>985</td>
</tr>
<tr>
<td>ToUpper</td>
<td>986</td>
</tr>
<tr>
<td>NAME</td>
<td>986</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>986</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>986</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>986</td>
</tr>
<tr>
<td>BUGS</td>
<td>987</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>987</td>
</tr>
<tr>
<td>WrongNumArgs</td>
<td>988</td>
</tr>
<tr>
<td>NAME</td>
<td>988</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>988</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>988</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>988</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>989</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>989</td>
</tr>
<tr>
<td>Thread</td>
<td>990</td>
</tr>
<tr>
<td>NAME</td>
<td>990</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>990</td>
</tr>
</tbody>
</table>
# Table of Contents

## Tcl Library

<table>
<thead>
<tr>
<th>NAME</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARGUMENTS</td>
<td>990</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>991</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>992</td>
</tr>
<tr>
<td>INITIALIZATION</td>
<td>993</td>
</tr>
<tr>
<td>CREATING THREADS</td>
<td>993</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>993</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>993</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>994</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>994</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>994</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>994</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>994</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>994</td>
</tr>
</tbody>
</table>

## Tk Library

<table>
<thead>
<tr>
<th>NAME</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>1001</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>1002</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1002</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>1003</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1003</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>1004</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1004</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>1005</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1005</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>1006</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1006</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>1008</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1008</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>1009</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1009</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>1010</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1010</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>1010</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1010</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>1011</td>
</tr>
<tr>
<td>BUGS</td>
<td>1011</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1014</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>1015</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1014</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>1015</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1015</td>
</tr>
</tbody>
</table>
# Table of Contents

## Tk Library

**SYNOPSIS** ................................................................................................................................. 1015
**ARGUMENTS** ............................................................................................................................... 1015
**DESCRIPTION** ............................................................................................................................. 1016
**KEYWORDS** ................................................................................................................................. 1018

### GetCursor

**NAME** ........................................................................................................................................ 1019
**SYNOPSIS** ................................................................................................................................ 1019
**ARGUMENTS** ............................................................................................................................... 1019
**DESCRIPTION** ............................................................................................................................. 1019
**BUGS** ......................................................................................................................................... 1020
**KEYWORDS** ................................................................................................................................ 1022

### GetFont

**NAME** ........................................................................................................................................ 1023
**SYNOPSIS** ................................................................................................................................ 1023
**ARGUMENTS** ............................................................................................................................... 1023
**DESCRIPTION** ............................................................................................................................. 1023
**SEE ALSO** .................................................................................................................................. 1024
**KEYWORDS** ................................................................................................................................ 1024

### CanvPsY

**NAME** ........................................................................................................................................ 1025
**SYNOPSIS** ................................................................................................................................ 1025
**ARGUMENTS** ............................................................................................................................... 1025
**DESCRIPTION** ............................................................................................................................. 1025

### CanvTxtInfo

**NAME** ........................................................................................................................................ 1028
**SYNOPSIS** ................................................................................................................................ 1028
**ARGUMENTS** ............................................................................................................................... 1028
**DESCRIPTION** ............................................................................................................................. 1028

### CanvTkwin

**NAME** ........................................................................................................................................ 1030
**SYNOPSIS** ................................................................................................................................ 1030
**ARGUMENTS** ............................................................................................................................... 1030
**DESCRIPTION** ............................................................................................................................. 1031

### ClearSelect

**NAME** ........................................................................................................................................ 1033
**SYNOPSIS** ................................................................................................................................ 1033
**ARGUMENTS** ............................................................................................................................... 1033
**DESCRIPTION** ............................................................................................................................. 1033

---

**Table of Contents**

- Tk Library
  - **SYNOPSIS** ................................................................................................................................. 1015
  - **ARGUMENTS** ............................................................................................................................... 1015
  - **DESCRIPTION** ............................................................................................................................. 1016
  - **KEYWORDS** ................................................................................................................................. 1018

- GetCursor
  - **NAME** ........................................................................................................................................ 1019
  - **SYNOPSIS** ................................................................................................................................ 1019
  - **ARGUMENTS** ............................................................................................................................... 1019
  - **DESCRIPTION** ............................................................................................................................. 1019
  - **BUGS** ......................................................................................................................................... 1020
  - **KEYWORDS** ................................................................................................................................ 1022

- GetFont
  - **NAME** ........................................................................................................................................ 1023
  - **SYNOPSIS** ................................................................................................................................ 1023
  - **ARGUMENTS** ............................................................................................................................... 1023
  - **DESCRIPTION** ............................................................................................................................. 1023
  - **SEE ALSO** .................................................................................................................................. 1024
  - **KEYWORDS** ................................................................................................................................ 1024

- CanvPsY
  - **NAME** ........................................................................................................................................ 1025
  - **SYNOPSIS** ................................................................................................................................ 1025
  - **ARGUMENTS** ............................................................................................................................... 1025
  - **DESCRIPTION** ............................................................................................................................. 1025

- CanvTxtInfo
  - **NAME** ........................................................................................................................................ 1028
  - **SYNOPSIS** ................................................................................................................................ 1028
  - **ARGUMENTS** ............................................................................................................................... 1028
  - **DESCRIPTION** ............................................................................................................................. 1028

- CanvTkwin
  - **NAME** ........................................................................................................................................ 1030
  - **SYNOPSIS** ................................................................................................................................ 1030
  - **ARGUMENTS** ............................................................................................................................... 1030
  - **DESCRIPTION** ............................................................................................................................. 1031

- ClearSelect
  - **NAME** ........................................................................................................................................ 1033
  - **SYNOPSIS** ................................................................................................................................ 1033
  - **ARGUMENTS** ............................................................................................................................... 1033
  - **DESCRIPTION** ............................................................................................................................. 1033
# Table of Contents

## Tk Library

<table>
<thead>
<tr>
<th>Subject</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clipboard</td>
<td>1034</td>
</tr>
<tr>
<td>NAME</td>
<td>1034</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1034</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1034</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1034</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1035</td>
</tr>
<tr>
<td>TextLayout</td>
<td>1036</td>
</tr>
<tr>
<td>NAME</td>
<td>1036</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1036</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1036</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1038</td>
</tr>
<tr>
<td>DISPLAY MODEL</td>
<td>1039</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1040</td>
</tr>
<tr>
<td>ConfigWidg</td>
<td>1041</td>
</tr>
<tr>
<td>NAME</td>
<td>1042</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1042</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1042</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1043</td>
</tr>
<tr>
<td>GROUPED ENTRIES</td>
<td>1046</td>
</tr>
<tr>
<td>FLAGS</td>
<td>1047</td>
</tr>
<tr>
<td>TK OFFSET</td>
<td>1048</td>
</tr>
<tr>
<td>TK CONFIGUREINFO</td>
<td>1048</td>
</tr>
<tr>
<td>TK CONFIGUREVALUE</td>
<td>1049</td>
</tr>
<tr>
<td>TK_FREEOPTIONS</td>
<td>1049</td>
</tr>
<tr>
<td>CUSTOM OPTION TYPES</td>
<td>1049</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>1050</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>1050</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1050</td>
</tr>
<tr>
<td>ConfigWind</td>
<td>1052</td>
</tr>
<tr>
<td>NAME</td>
<td>1052</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1052</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1052</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1053</td>
</tr>
<tr>
<td>BUGS</td>
<td>1054</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>1054</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1054</td>
</tr>
<tr>
<td>CoordToWin</td>
<td>1055</td>
</tr>
<tr>
<td>NAME</td>
<td>1055</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1055</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1055</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1055</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1055</td>
</tr>
</tbody>
</table>
# Table of Contents

## Tk Library

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BindTable</td>
<td>1056</td>
</tr>
<tr>
<td>NAME</td>
<td>1056</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1056</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1056</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1057</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1058</td>
</tr>
<tr>
<td>CrtCmHdlr</td>
<td>1059</td>
</tr>
<tr>
<td>NAME</td>
<td>1059</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1059</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1059</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1059</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1060</td>
</tr>
<tr>
<td>CrtErrHdlr</td>
<td>1061</td>
</tr>
<tr>
<td>NAME</td>
<td>1061</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1061</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1061</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1061</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1063</td>
</tr>
<tr>
<td>EventHndlr</td>
<td>1064</td>
</tr>
<tr>
<td>NAME</td>
<td>1064</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1064</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1064</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1064</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1065</td>
</tr>
<tr>
<td>CrtGenHdlr</td>
<td>1066</td>
</tr>
<tr>
<td>NAME</td>
<td>1066</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1066</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1066</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1066</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1067</td>
</tr>
<tr>
<td>CrtImgType</td>
<td>1068</td>
</tr>
<tr>
<td>NAME</td>
<td>1068</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1068</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1068</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1069</td>
</tr>
<tr>
<td>NAME</td>
<td>1070</td>
</tr>
<tr>
<td>PORTABILITY</td>
<td>1070</td>
</tr>
<tr>
<td>CREATEPROC</td>
<td>1070</td>
</tr>
<tr>
<td>GETPROC</td>
<td>1071</td>
</tr>
<tr>
<td>DISPLAYPROC</td>
<td>1071</td>
</tr>
<tr>
<td>FREEPROC</td>
<td>1071</td>
</tr>
<tr>
<td>DELETEPROC</td>
<td>1072</td>
</tr>
</tbody>
</table>
Table of Contents

Tk Library

TK_GETIMAGEMASTERDATA ................................................................. 1072
TK_INITIMAGEARGS ................................................................ 1072
SEE ALSO ............................................................................. 1072
KEYWORDS .......................................................................... 1073

CrtItemType .................................................................................. 1074
NAME .................................................................................. 1074
SYNOPSIS ............................................................................ 1074
ARGUMENTS ........................................................................ 1075
INTRODUCTION ..................................................................... 1075
DATA STRUCTURES .................................................................. 1075
NAME ............................................................................... 1077
ITEMSIZE ............................................................................ 1077
CREATEPROC .......................................................................... 1077
CONFIGSPECS ....................................................................... 1077
CONFIGPROC .......................................................................... 1078
COORDPROC .......................................................................... 1078
DELETEPROC .......................................................................... 1079
DISPLAYPROC AND ALWAYSREDRAW ........................................ 1079
POINTPROC .......................................................................... 1080
AREAPROC ........................................................................... 1080
POSTSCRIPTPROC ................................................................... 1080
SCALEPROC ............................................................................ 1081
TRANSLATEPROC ................................................................... 1082
INDEXPROC ........................................................................... 1082
ICURSORPROC ....................................................................... 1082
SELECTIONPROC .................................................................... 1083
INSERTPROC ........................................................................ 1083
DCCHARSPROC ....................................................................... 1083
SEE ALSO ............................................................................ 1084
KEYWORDS ............................................................................ 1084

CrtPhlImgFmt .................................................................................. 1085
NAME .................................................................................. 1085
SYNOPSIS ............................................................................ 1085
ARGUMENTS ........................................................................ 1085
DESCRIPTION ....................................................................... 1085
PORTABILITY .......................................................................... 1086
NAME ............................................................................... 1086
FILEMATCHPROC ................................................................... 1086
STRINGMATCHPROC .................................................................. 1087
FILEREADPROC ....................................................................... 1087
STRINGREADPROC .................................................................. 1088
FILEWRITEPROC .................................................................... 1088
# Table of Contents

**Tk Library**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRINGWRITEPROC</td>
<td>1088</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>1089</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1089</td>
</tr>
<tr>
<td>CrtSelHdr</td>
<td>1090</td>
</tr>
<tr>
<td>NAME</td>
<td>1090</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1090</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1090</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1090</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1091</td>
</tr>
<tr>
<td>CrtWindow</td>
<td>1092</td>
</tr>
<tr>
<td>NAME</td>
<td>1092</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1092</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1092</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1092</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1094</td>
</tr>
<tr>
<td>Deleting</td>
<td>1095</td>
</tr>
<tr>
<td>NAME</td>
<td>1095</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1095</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1095</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1095</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1095</td>
</tr>
<tr>
<td>DrawFocHit</td>
<td>1096</td>
</tr>
<tr>
<td>NAME</td>
<td>1096</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1096</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1096</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1096</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1096</td>
</tr>
<tr>
<td>FindPhoto</td>
<td>1097</td>
</tr>
<tr>
<td>NAME</td>
<td>1097</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1097</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1097</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1098</td>
</tr>
<tr>
<td>PORTABILITY</td>
<td>1100</td>
</tr>
<tr>
<td>CREDITS</td>
<td>1100</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1100</td>
</tr>
<tr>
<td>FontId</td>
<td>1101</td>
</tr>
<tr>
<td>NAME</td>
<td>1101</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1101</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1101</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1101</td>
</tr>
<tr>
<td>DATA STRUCTURES</td>
<td>1102</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1102</td>
</tr>
</tbody>
</table>
# Table of Contents

## Tk Library

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeXId</td>
<td>1103</td>
</tr>
<tr>
<td>NAME</td>
<td>1103</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1103</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1103</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1103</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1103</td>
</tr>
<tr>
<td>GeomReq</td>
<td>1104</td>
</tr>
<tr>
<td>NAME</td>
<td>1104</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1104</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1104</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1104</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1105</td>
</tr>
<tr>
<td>GetAnchor</td>
<td>1106</td>
</tr>
<tr>
<td>NAME</td>
<td>1106</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1106</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1106</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1106</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1107</td>
</tr>
<tr>
<td>GetCapStyl</td>
<td>1108</td>
</tr>
<tr>
<td>NAME</td>
<td>1108</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1108</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1108</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1108</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1108</td>
</tr>
<tr>
<td>GetClrman</td>
<td>1110</td>
</tr>
<tr>
<td>NAME</td>
<td>1110</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1110</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1110</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1110</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1111</td>
</tr>
<tr>
<td>GetDash</td>
<td>1112</td>
</tr>
<tr>
<td>NAME</td>
<td>1112</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1112</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1112</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1112</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1113</td>
</tr>
<tr>
<td>GetGC</td>
<td>1114</td>
</tr>
<tr>
<td>NAME</td>
<td>1114</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1114</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1114</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1114</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1115</td>
</tr>
</tbody>
</table>
# Table of Contents

## Tk Library

<table>
<thead>
<tr>
<th>Function</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetHINSTANCE</td>
<td>1116</td>
</tr>
<tr>
<td>NAME</td>
<td>1116</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1116</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1116</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1116</td>
</tr>
<tr>
<td>GetImage</td>
<td>1117</td>
</tr>
<tr>
<td>NAME</td>
<td>1117</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1117</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1117</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1118</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>1119</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1119</td>
</tr>
<tr>
<td>GetJoinStl</td>
<td>1120</td>
</tr>
<tr>
<td>NAME</td>
<td>1120</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1120</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1120</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1120</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1120</td>
</tr>
<tr>
<td>GetJustify</td>
<td>1122</td>
</tr>
<tr>
<td>NAME</td>
<td>1122</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1122</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1122</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1123</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1123</td>
</tr>
<tr>
<td>GetOption</td>
<td>1124</td>
</tr>
<tr>
<td>NAME</td>
<td>1124</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1124</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1124</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1124</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1124</td>
</tr>
<tr>
<td>GetPixels</td>
<td>1125</td>
</tr>
<tr>
<td>NAME</td>
<td>1125</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1125</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1125</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1126</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1126</td>
</tr>
<tr>
<td>GetPixmap</td>
<td>1127</td>
</tr>
<tr>
<td>NAME</td>
<td>1127</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1127</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1127</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1127</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1127</td>
</tr>
</tbody>
</table>
# Table of Contents

## Tk Library

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetRelief</td>
<td>1129</td>
</tr>
<tr>
<td>NAME</td>
<td>1129</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1129</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1129</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1129</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1130</td>
</tr>
<tr>
<td>GetRootCrd</td>
<td>1131</td>
</tr>
<tr>
<td>NAME</td>
<td>1131</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1131</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1131</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1131</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1131</td>
</tr>
<tr>
<td>GetScroll</td>
<td>1132</td>
</tr>
<tr>
<td>NAME</td>
<td>1132</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1132</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1132</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1132</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1133</td>
</tr>
<tr>
<td>GetSelect</td>
<td>1134</td>
</tr>
<tr>
<td>NAME</td>
<td>1134</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1134</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1134</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1134</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1135</td>
</tr>
<tr>
<td>GetUid</td>
<td>1136</td>
</tr>
<tr>
<td>NAME</td>
<td>1136</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1136</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1136</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1136</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1136</td>
</tr>
<tr>
<td>GetVRoot</td>
<td>1137</td>
</tr>
<tr>
<td>NAME</td>
<td>1137</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1137</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1137</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1137</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1137</td>
</tr>
<tr>
<td>GetVisual</td>
<td>1138</td>
</tr>
<tr>
<td>NAME</td>
<td>1138</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1138</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1138</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1138</td>
</tr>
<tr>
<td>CREDITS</td>
<td>1139</td>
</tr>
</tbody>
</table>
# Table of Contents

## Tk Library

<table>
<thead>
<tr>
<th>Function</th>
<th>NAME</th>
<th>SYNOPSIS</th>
<th>ARGUMENTS</th>
<th>DESCRIPTION</th>
<th>KEYWORDS</th>
<th>KEYWORDS</th>
<th>DESCRIPTION</th>
<th>ARGUMENTS</th>
<th>SYNOPSIS</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>TkInitStubs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TkInit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TkInitStubs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Grab</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HandleEvent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HWNDToWindow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IdToWindow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ImgChanged</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Table of Contents

## Tk Library

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>1149</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1150</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>1150</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1150</td>
</tr>
<tr>
<td>InternAtom</td>
<td>1151</td>
</tr>
<tr>
<td>NAME</td>
<td>1151</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1151</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1151</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1151</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1151</td>
</tr>
<tr>
<td>Tk Main</td>
<td>1152</td>
</tr>
<tr>
<td>NAME</td>
<td>1152</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1152</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1152</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1152</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1153</td>
</tr>
<tr>
<td>MainLoop</td>
<td>1154</td>
</tr>
<tr>
<td>NAME</td>
<td>1154</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1154</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1154</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1154</td>
</tr>
<tr>
<td>MainWin</td>
<td>1155</td>
</tr>
<tr>
<td>NAME</td>
<td>1155</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1155</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1155</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1155</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1155</td>
</tr>
<tr>
<td>MaintGeom</td>
<td>1156</td>
</tr>
<tr>
<td>NAME</td>
<td>1156</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1156</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1156</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1156</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1157</td>
</tr>
<tr>
<td>ManageGeom</td>
<td>1158</td>
</tr>
<tr>
<td>NAME</td>
<td>1158</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1158</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>1158</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>1158</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>1159</td>
</tr>
<tr>
<td>MapWindow</td>
<td>1160</td>
</tr>
<tr>
<td>NAME</td>
<td>1160</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>1160</td>
</tr>
</tbody>
</table>

- **SEE ALSO**

- **INTRODUCTION**

- **DESCRIPTION**

- **ARGUMENTS**

- **SYNOPSIS**
# Table of Contents

## Tk Library

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParseArgv</td>
<td>1169</td>
</tr>
<tr>
<td>OwnSelect</td>
<td>1168</td>
</tr>
<tr>
<td>MeasureChar</td>
<td>1161</td>
</tr>
<tr>
<td>Name</td>
<td>1165</td>
</tr>
<tr>
<td>NameOfImg</td>
<td>1167</td>
</tr>
<tr>
<td>QWinEvent</td>
<td>1176</td>
</tr>
</tbody>
</table>

---

**ARGUMENTS** .................................................................................................................................1160
**DESCRIPTION** .................................................................................................................................1160
**KEYWORDS** .........................................................................................................................................1160

**NAME** ..............................................................................................................................................1161
**SYNOPSIS** .........................................................................................................................................1161
**ARGUMENTS** .......................................................................................................................................1161
**DESCRIPTION** ...................................................................................................................................1162
**KEYWORDS** .........................................................................................................................................1163

**NAME** ..............................................................................................................................................1164
**SYNOPSIS** .........................................................................................................................................1164
**ARGUMENTS** .......................................................................................................................................1164
**DESCRIPTION** ...................................................................................................................................1164
**KEYWORDS** .........................................................................................................................................1164

**NAME** ..............................................................................................................................................1165
**SYNOPSIS** .........................................................................................................................................1165
**ARGUMENTS** .......................................................................................................................................1165
**DESCRIPTION** ...................................................................................................................................1165
**KEYWORDS** .........................................................................................................................................1166

**NAME** ..............................................................................................................................................1167
**SYNOPSIS** .........................................................................................................................................1167
**ARGUMENTS** .......................................................................................................................................1167
**DESCRIPTION** ...................................................................................................................................1167
**KEYWORDS** .........................................................................................................................................1167

**NAME** ..............................................................................................................................................1168
**SYNOPSIS** .........................................................................................................................................1168
**ARGUMENTS** .......................................................................................................................................1168
**DESCRIPTION** ...................................................................................................................................1168
**KEYWORDS** .........................................................................................................................................1168

**NAME** ..............................................................................................................................................1169
**SYNOPSIS** .........................................................................................................................................1169
**ARGUMENTS** .......................................................................................................................................1169
**DESCRIPTION** ...................................................................................................................................1170
**FLAGS** ...............................................................................................................................................1173
**EXAMPLE** .........................................................................................................................................1173
**KEYWORDS** .........................................................................................................................................1174
# Table of Contents

## Tk Library

**NAME** ...............................................................................................................................................1176
**SYNOPSIS** .......................................................................................................................................1176
**ARGUMENTS** ....................................................................................................................................1176
**DESCRIPTION** ..............................................................................................................................1176
**KEYWORDS** ......................................................................................................................................1176

**Restrack** ...........................................................................................................................................1177
**NAME** .............................................................................................................................................1177
**SYNOPSIS** ........................................................................................................................................1177
**ARGUMENTS** .....................................................................................................................................1177
**DESCRIPTION** ..............................................................................................................................1177
**KEYWORDS** ......................................................................................................................................1177

**RestrictEv** .......................................................................................................................................1178
**NAME** .............................................................................................................................................1178
**SYNOPSIS** ........................................................................................................................................1178
**ARGUMENTS** .....................................................................................................................................1178
**DESCRIPTION** ..............................................................................................................................1178
**KEYWORDS** ......................................................................................................................................1179

**SetAppName** .....................................................................................................................................1180
**NAME** .............................................................................................................................................1180
**SYNOPSIS** ........................................................................................................................................1180
**ARGUMENTS** .....................................................................................................................................1180
**DESCRIPTION** ..............................................................................................................................1180
**KEYWORDS** ......................................................................................................................................1181

**SetCaret** ..........................................................................................................................................1182
**NAME** .............................................................................................................................................1182
**SYNOPSIS** ........................................................................................................................................1182
**ARGUMENTS** .....................................................................................................................................1182
**DESCRIPTION** ..............................................................................................................................1182
**KEYWORDS** ......................................................................................................................................1182

**SetClass** ..........................................................................................................................................1183
**NAME** .............................................................................................................................................1183
**SYNOPSIS** ........................................................................................................................................1183
**ARGUMENTS** .....................................................................................................................................1183
**DESCRIPTION** ..............................................................................................................................1183
**KEYWORDS** ......................................................................................................................................1183

**SetClassProcs** ...............................................................................................................................1184
**NAME** .............................................................................................................................................1184
**SYNOPSIS** ........................................................................................................................................1184
**ARGUMENTS** .....................................................................................................................................1184
**DESCRIPTION** ..............................................................................................................................1184
**KEYWORDS** ......................................................................................................................................1185

**SetGrid** ............................................................................................................................................1186
Table of Contents

Tk Library

NAME .................................................................................................................................................1186
SYNOPSIS .................................................................................................................................1186
ARGUMENTS ..............................................................................................................................1186
DESCRIPTION ............................................................................................................................1186
KEYWORDS ...................................................................................................................................1187

SetOptions

NAME .............................................................................................................................................1188
SYNOPSIS .................................................................................................................................1189
ARGUMENTS ..............................................................................................................................1189
DESCRIPTION ............................................................................................................................1189
TEMPLATES ..............................................................................................................................1190
STORAGE MANAGEMENT ISSUES ..........................................................................................1192
OBJOFFSET VS. INTERNALOFFSET ......................................................................................1195
CUSTOM OPTION TYPES .......................................................................................................1196
KEYWORDS ...................................................................................................................................1196

SetVisual

NAME .............................................................................................................................................1199
SYNOPSIS .................................................................................................................................1199
ARGUMENTS ..............................................................................................................................1199
DESCRIPTION ............................................................................................................................1199
KEYWORDS ...................................................................................................................................1199

StrictMotif

NAME .............................................................................................................................................1200
SYNOPSIS .................................................................................................................................1200
ARGUMENTS ..............................................................................................................................1200
DESCRIPTION ............................................................................................................................1200
KEYWORDS ...................................................................................................................................1200

WindowId

NAME .............................................................................................................................................1201
SYNOPSIS .................................................................................................................................1201
ARGUMENTS ..............................................................................................................................1202
DESCRIPTION ............................................................................................................................1202
KEYWORDS ...................................................................................................................................1203

Tcl/Tk Keywords

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z ...................................................................1204
Tcl/Tk Keywords – A ..................................................................................................................1205
Tcl/Tk Keywords – B ..................................................................................................................1207
Tcl/Tk Keywords – C ..................................................................................................................1209
Tcl/Tk Keywords – D ..................................................................................................................1213
Tcl/Tk Keywords – E ..................................................................................................................1215
Tcl/Tk Keywords – F ..................................................................................................................1217
# Table of Contents

**Tcl/Tk Keywords**

- Tcl/Tk Keywords – G ................................................................. 1219
- Tcl/Tk Keywords – H ................................................................. 1221
- Tcl/Tk Keywords – I ................................................................. 1222
- Tcl/Tk Keywords – J ................................................................. 1225
- Tcl/Tk Keywords – K ................................................................. 1226
- Tcl/Tk Keywords – L ................................................................. 1227
- Tcl/Tk Keywords – M ................................................................. 1229
- Tcl/Tk Keywords – N ................................................................. 1231
- Tcl/Tk Keywords – O ................................................................. 1233
- Tcl/Tk Keywords – P ................................................................. 1235
- Tcl/Tk Keywords – Q ................................................................. 1238
- Tcl/Tk Keywords – R ................................................................. 1239
- Tcl/Tk Keywords – S ................................................................. 1242
- Tcl/Tk Keywords – T ................................................................. 1246
- Tcl/Tk Keywords – U ................................................................. 1249
- Tcl/Tk Keywords – V ................................................................. 1251
- Tcl/Tk Keywords – W ................................................................. 1253
- Tcl/Tk Keywords – X ................................................................. 1255
- Tcl/Tk Keywords – Y ................................................................. 1256
- Tcl/Tk Keywords – Z ................................................................. 1257
Tcl8.4/Tk8.4 Manual

Tcl/Tk Applications
The interpreters which implement Tcl and Tk.

Tcl Commands
The commands which the tclsh interpreter implements.

Tk Commands
The additional commands which the wish interpreter implements.

Tcl Library
The C functions which a Tcl extended C program may use.

Tk Library
The additional C functions which a Tk extended C program may use.

Keywords
The keywords from the Tcl/Tk man pages.

Source
More information about these man pages.

Copyright ' 1989−1994 The Regents of the University of California.
Copyright ' 1990−1994 The Regents of the University of California
Copyright ' 1992−1999 Karl Lehenbauer and Mark Diekhans
Copyright ' 1992−1999 Karl Lehenbauer and Mark Diekhans.
Copyright ' 1993−1997 Bell Labs Innovations for Lucent Technologies
Copyright ' 1994 The Australian National University
Copyright ' 1994−2000 Sun Microsystems, Inc.
Copyright ' 1997−2000 Ajuba Solutions.
Copyright ' 1997−2000 Scriptics Corporation.
Copyright ' 1998 Mark Harrison.
Copyright ' 1998−1999 Scriptics Corporation
Copyright ' 1998−2000 Ajuba Solutions
Copyright ' 1999 Scriptics Corporation
Copyright ' 2000 Jeffrey Hobbs.
Copyright ' 2001 ActiveState Tool Corp.
Copyright ' 2001 Kevin B. Kenny.
Copyright ' 2001 Vincent Darley
Copyright ' 2001−2002 ActiveState Corporation
Copyright ' 2001−2002 Donal K. Fellows
Copyright ' 2001−2002 Kevin B. Kenny. All rights reserved.
Copyright ' 2001−2004 ActiveState Corporation.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
Tcl/Tk Applications

tclsh  wish

Copyright  ' 1991-1994 The Regents of the University of California.
Copyright  ' 1994-1996 Sun Microsystems, Inc.
Copyright  ' 1995-1997 Roger E. Critchlow Jr.
NAME

tclsh – Simple shell containing Tcl interpreter

SYNOPSIS

tclsh ?

tclsh ?fileName arg arg ...?

DESCRIPTION

Tclsh is a shell−like application that reads Tcl commands from its standard input or from a file and evaluates them. If invoked with no arguments then it runs interactively, reading Tcl commands from standard input and printing command results and error messages to standard output. It runs until the exit command is invoked or until it reaches end−of−file on its standard input. If there exists a file .tclshrc (or tclshrc.tcl on the Windows platforms) in the home directory of the user, tclsh evaluates the file as a Tcl script just before reading the first command from standard input.

SCRIPT FILES

If tclsh is invoked with arguments then the first argument is the name of a script file and any additional arguments are made available to the script as variables (see below). Instead of reading commands from standard input tclsh will read Tcl commands from the named file; tclsh will exit when it reaches the end of the file. The end of the file may be marked either by the physical end of the medium, or by the character, \032 (\u001a, control−Z). If this character is present in the file, the tclsh application will read text up to but
not including the character. An application that requires this character in the file may safely encode it as `\032`, `\x1a`, or `\u001a`; or may generate it by use of commands such as `format` or `binary`. There is no automatic evaluation of `.tclshrc` when the name of a script file is presented on the `tclsh` command line, but the script file can always `source` it if desired.

If you create a Tcl script in a file whose first line is

```bash
#!/usr/local/bin/tclsh
```

then you can invoke the script file directly from your shell if you mark the file as executable. This assumes that `tclsh` has been installed in the default location in `/usr/local/bin`; if it's installed somewhere else then you'll have to modify the above line to match. Many UNIX systems do not allow the `#!` line to exceed about 30 characters in length, so be sure that the `tclsh` executable can be accessed with a short file name.

An even better approach is to start your script files with the following three lines:

```bash
#!/bin/sh
# the next line restarts using tclsh 
exec tclsh "$0" "$@"
```

This approach has three advantages over the approach in the previous paragraph. First, the location of the `tclsh` binary doesn't have to be hard–wired into the script: it can be anywhere in your shell search path. Second, it gets around the 30–character file name limit in the previous approach. Third, this approach will work even if `tclsh` is itself a shell script (this is done on some systems in order to handle multiple architectures or operating systems: the `tclsh` script selects one of several binaries to run). The three lines cause both `sh` and `tclsh` to process the script, but the `exec` is only executed by `sh`. `sh` processes the script first; it treats the second line as a comment and executes the third line. The `exec` statement cause the shell to stop processing and instead to start up `tclsh` to reprocess the entire script. When `tclsh` starts up, it treats all three lines as comments, since the backslash at the end of the second line causes the third line to be treated as part of the comment on the second line.

You should note that it is also common practise to install `tclsh` with its version number as part of the name. This has the advantage of allowing multiple versions of Tcl to exist on the same system at once, but also the disadvantage of making it harder to write scripts that start up uniformly across different versions of Tcl.

**VARIABLES**

`Tclsh` sets the following Tcl variables:

- `argc`: Contains a count of the number of `arg` arguments (0 if none), not including the name of the script file.
- `argv`: Contains a Tcl list whose elements are the `arg` arguments, in order, or an empty string if there are no `arg` arguments.
- `argv0`: Contains `fileName` if it was specified. Otherwise, contains the name by which `tclsh` was invoked.
**tcl_interactive**

Contains 1 if *tclsh* is running interactively (no *fileName* was specified and standard input is a terminal−like device), 0 otherwise.

**PROMPTS**

When *tclsh* is invoked interactively it normally prompts for each command with `` % ''. You can change the prompt by setting the variables *tcl_prompt1* and *tcl_prompt2*. If variable *tcl_prompt1* exists then it must consist of a Tcl script to output a prompt; instead of outputting a prompt *tclsh* will evaluate the script in *tcl_prompt1*. The variable *tcl_prompt2* is used in a similar way when a newline is typed but the current command isn't yet complete; if *tcl_prompt2* isn't set then no prompt is output for incomplete commands.

**STANDARD CHANNELS**

See [Tcl StandardChannels](#) for more explanations.

**SEE ALSO**

- `fconfigure`, `tclvars`

**KEYWORDS**

- argument, interpreter, prompt, script file, shell

---

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994−1996 Sun Microsystems, Inc.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
NAME

wish – Simple windowing shell

SYNOPSIS

wish ?fileName arg arg ...?

OPTIONS

−colormap new
−display display
−geometry geometry
−name name
−sync
−use id
−visual visual

DESCRIPTION

APPLICATION NAME AND CLASS

VARIABLES

argc
argv
argv0
geometry
tcl_interactive

SCRIPT FILES

PROMPTS

KEYWORDS

NAME

wish – Simple windowing shell

SYNOPSIS

wish ?fileName arg arg ...?

OPTIONS

−colormap new
   Specifies that the window should have a new private colormap instead of using the default colormap for the screen.
−display display
Display (and screen) on which to display window.

\texttt{−geometry geometry}

Initial geometry to use for window. If this option is specified, its value is stored in the \texttt{geometry} global variable of the application's Tcl interpreter.

\texttt{−name name}

Use \texttt{name} as the title to be displayed in the window, and as the name of the interpreter for \texttt{send} commands.

\texttt{−sync}

Execute all X server commands synchronously, so that errors are reported immediately. This will result in much slower execution, but it is useful for debugging.

\texttt{−use id}

Specifies that the main window for the application is to be embedded in the window whose identifier is \texttt{id}, instead of being created as an independent toplevel window. \texttt{Id} must be specified in the same way as the value for the \texttt{−use} option for toplevel widgets (i.e. it has a form like that returned by the \texttt{winfo id} command).

\texttt{−visual visual}

Specifies the visual to use for the window. \texttt{Visual} may have any of the forms supported by the \texttt{Tk GetVisual} procedure.

\texttt{− −}

Pass all remaining arguments through to the script's \texttt{argv} variable without interpreting them. This provides a mechanism for passing arguments such as \texttt{−name} to a script instead of having \texttt{wish} interpret them.

\section*{DESCRIPTION}

\textbf{Wish} is a simple program consisting of the Tcl command language, the Tk toolkit, and a main program that reads commands from standard input or from a file. It creates a main window and then processes Tcl commands. If \texttt{wish} is invoked with no arguments, or with a first argument that starts with `−−', then it reads Tcl commands interactively from standard input. It will continue processing commands until all windows have been deleted or until end-of-file is reached on standard input. If there exists a file \texttt{.wishrc} in the home directory of the user, \texttt{wish} evaluates the file as a Tcl script just before reading the first command from standard input.

If \texttt{wish} is invoked with an initial \texttt{fileName} argument, then \texttt{fileName} is treated as the name of a script file. \textbf{Wish} will evaluate the script in \texttt{fileName} (which presumably creates a user interface), then it will respond to events until all windows have been deleted. Commands will not be read from standard input. There is no automatic evaluation of \texttt{.wishrc} when the name of a script file is presented on the \texttt{wish} command line, but the script file can always \texttt{source} it if desired.

\section*{OPTIONS}

\textbf{Wish} automatically processes all of the command-line options described in the \texttt{OPTIONS} summary above. Any other command-line arguments besides these are passed through to the application using the \texttt{argc} and \texttt{argv} variables described later.
APPLICATION NAME AND CLASS

The name of the application, which is used for purposes such as send commands, is taken from the −name option, if it is specified; otherwise it is taken from fileName, if it is specified, or from the command name by which wish was invoked. In the last two cases, if the name contains a "/" character, then only the characters after the last slash are used as the application name.

The class of the application, which is used for purposes such as specifying options with a RESOURCE_MANAGER property or .Xdefaults file, is the same as its name except that the first letter is capitalized.

VARIABLES

Wish sets the following Tcl variables:

argc
Contains a count of the number of arg arguments (0 if none), not including the options described above.

argv
Contains a Tcl list whose elements are the arg arguments that follow a − − option or don't match any of the options described in OPTIONS above, in order, or an empty string if there are no such arguments.

argv0
Contains fileName if it was specified. Otherwise, contains the name by which wish was invoked.

geometry
If the −geometry option is specified, wish copies its value into this variable. If the variable still exists after fileName has been evaluated, wish uses the value of the variable in a wm geometry command to set the main window's geometry.

tcl_interactive
Contains 1 if wish is reading commands interactively (fileName was not specified and standard input is a terminal–like device), 0 otherwise.

SCRIPT FILES

If you create a Tcl script in a file whose first line is

#!/usr/local/bin/wish

then you can invoke the script file directly from your shell if you mark it as executable. This assumes that wish has been installed in the default location in /usr/local/bin; if it's installed somewhere else then you'll have to modify the above line to match. Many UNIX systems do not allow the #! line to exceed about 30 characters in length, so be sure that the wish executable can be accessed with a short file name.

An even better approach is to start your script files with the following three lines:
#!/bin/sh
# the next line restarts using wish 
exec wish "$0" "$@"

This approach has three advantages over the approach in the previous paragraph. First, the location of the wish binary doesn't have to be hard-wired into the script: it can be anywhere in your shell search path. Second, it gets around the 30-character file name limit in the previous approach. Third, this approach will work even if wish is itself a shell script (this is done on some systems in order to handle multiple architectures or operating systems: the wish script selects one of several binaries to run). The three lines cause both sh and wish to process the script, but the exec is only executed by sh. sh processes the script first; it treats the second line as a comment and executes the third line. The exec statement cause the shell to stop processing and instead to start up wish to reprocess the entire script. When wish starts up, it treats all three lines as comments, since the backslash at the end of the second line causes the third line to be treated as part of the comment on the second line.

The end of a script file may be marked either by the physical end of the medium, or by the character, \032 ('\u001a', control-Z). If this character is present in the file, the wish application will read text up to but not including the character. An application that requires this character in the file may encode it as ``\032'', ``\x1a'', or ``\u001a''; or may generate it by use of commands such as format or binary.

**PROMPTS**

When wish is invoked interactively it normally prompts for each command with ``% ''. You can change the prompt by setting the variables tcl_prompt1 and tcl_prompt2. If variable tcl_prompt1 exists then it must consist of a Tcl script to output a prompt; instead of outputting a prompt wish will evaluate the script in tcl_prompt1. The variable tcl_prompt2 is used in a similar way when a newline is typed but the current command isn't yet complete; if tcl_prompt2 isn't set then no prompt is output for incomplete commands.

**KEYWORDS**

shell, toolkit

Copyright ' 1991-1994 The Regents of the University of California.  
Copyright ' 1994-1996 Sun Microsystems, Inc.  
Copyright ' 1995-1997 Roger E. Critchlow Jr.
<table>
<thead>
<tr>
<th>Safe Base</th>
<th>encoding</th>
<th>if</th>
<th>pid</th>
<th>switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcl</td>
<td>eof</td>
<td>incr</td>
<td>pkg::create</td>
<td>tcl_endOfWord</td>
</tr>
<tr>
<td>after</td>
<td>error</td>
<td>info</td>
<td>pkg_mkIndex</td>
<td>tcl_findLibrary</td>
</tr>
<tr>
<td>append</td>
<td>eval</td>
<td>interp</td>
<td>proc</td>
<td>tcl_startOfNextWord</td>
</tr>
<tr>
<td>array</td>
<td>exec</td>
<td>join</td>
<td>puts</td>
<td>tcl_startOfPreviousWord</td>
</tr>
<tr>
<td>auto_execok</td>
<td>exit</td>
<td>lappend</td>
<td>pwd</td>
<td>tcl_wordBreakAfter</td>
</tr>
<tr>
<td>auto_import</td>
<td>expr</td>
<td>lindex</td>
<td>re_syntax</td>
<td>tcl_wordBreakBefore</td>
</tr>
<tr>
<td>auto_load</td>
<td>fblocked</td>
<td>linsert</td>
<td>read</td>
<td>teltest</td>
</tr>
<tr>
<td>auto_mkindex</td>
<td>fconfigure</td>
<td>list</td>
<td>regexp</td>
<td>tclvars</td>
</tr>
<tr>
<td>auto_mkindex_old</td>
<td>fcopy</td>
<td>length</td>
<td>registry</td>
<td>tell</td>
</tr>
<tr>
<td>auto_qualify</td>
<td>file</td>
<td>load</td>
<td>regsub</td>
<td>time</td>
</tr>
<tr>
<td>auto_reset</td>
<td>fileevent</td>
<td>lrange</td>
<td>rename</td>
<td>trace</td>
</tr>
<tr>
<td>bgerror</td>
<td>filename</td>
<td>lreplace</td>
<td>resource</td>
<td>unknown</td>
</tr>
<tr>
<td>binary</td>
<td>flush</td>
<td>lsearch</td>
<td>return</td>
<td>unset</td>
</tr>
<tr>
<td>break</td>
<td>for</td>
<td>lset</td>
<td>scan</td>
<td>update</td>
</tr>
<tr>
<td>catch</td>
<td>foreach</td>
<td>lsort</td>
<td>seek</td>
<td>uplevel</td>
</tr>
<tr>
<td>cd</td>
<td>format</td>
<td>memory</td>
<td>set</td>
<td>upvar</td>
</tr>
<tr>
<td>clock</td>
<td>gets</td>
<td>msgcat</td>
<td>socket</td>
<td>variable</td>
</tr>
<tr>
<td>close</td>
<td>glob</td>
<td>namespace</td>
<td>source</td>
<td>vwait</td>
</tr>
<tr>
<td>concat</td>
<td>global</td>
<td>open</td>
<td>split</td>
<td>while</td>
</tr>
<tr>
<td>continue</td>
<td>history</td>
<td>package</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>dde</td>
<td>http</td>
<td>pararray</td>
<td>subst</td>
<td></td>
</tr>
</tbody>
</table>

Copyright ' 1990−1994 The Regents of the University of California
Copyright ' 1990−1994 The Regents of the University of California.
Copyright ' 1992−1995 Karl Lehenbauer and Mark Diekhans.
Copyright ' 1992−1999 Karl Lehenbauer and Mark Diekhans
Copyright ' 1993−1997 Bell Labs Innovations for Lucent Technologies
Copyright ' 1994−2000 Sun Microsystems, Inc.
Copyright ' 1998 Mark Harrison.
Copyright ' 1998−1999 Scriptics Corporation
Copyright ' 1998−2000 Ajuba Solutions
Copyright ' 1998−2000 Ajuba Solutions.
Copyright ' 2001 Donal K. Fellows
Copyright ' 2001 Kevin B. Kenny. All rights reserved.
Copyright ' 2001−2004 ActiveState Corporation.
Copyright ' 2002 ActiveState Corporation
Copyright ' 1995−1997 Roger E. Critchlow Jr.

Tcl Commands
NAME

Safe Base – A mechanism for creating and manipulating safe interpreters.

SYNOPSIS
::safe::interpCreate ?slave? ?options...?
::safe::interpInit slave ?options...?
::safe::interpConfigure slave ?options...?
::safe::interpDelete slave
::safe::interpAddToAccessPath slave directory
::safe::interpFindInAccessPath slave directory
::safe::setLogCmd ?cmd arg...?

OPTIONS

DESCRIPTION

COMMANDS
::safe::interpCreate ?slave? ?options...?
::safe::interpInit slave ?options...?
::safe::interpConfigure slave ?options...?
::safe::interpDelete slave
::safe::interpAddToAccessPath slave directory
::safe::interpFindInAccessPath slave directory
::safe::setLogCmd ?cmd arg...?

OPTIONS

-accessPath directoryList
-statics boolean
-noStatics
-nested boolean
-nestedLoadOk
-deleteHook script

ALIASES

source fileName
load fileName
file ?subCmd args...?
encoding ?subCmd args...?
exit

SECURITY

SEE ALSO

KEYWORDS
NAME

Safe Base – A mechanism for creating and manipulating safe interpreters.

SYNOPSIS

::safe::interpCreate ?slave? ?options...?
::safe::interpInit slave ?options...?
::safe::interpConfigure slave ?options...?
::safe::interpDelete slave
::safe::interpAddToAccessPath slave directory
::safe::interpFindInAccessPath slave directory
::safe::setLogCmd ?cmd arg...?

OPTIONS

?–accessPath pathList? ?–statics boolean? ?–noStatics? ?–nested boolean? ?–nestedLoadOk?
?–deleteHook script?

DESCRIPTION

Safe Tcl is a mechanism for executing untrusted Tcl scripts safely and for providing mediated access by such
scripts to potentially dangerous functionality.

The Safe Base ensures that untrusted Tcl scripts cannot harm the hosting application. The Safe Base prevents
integrity and privacy attacks. Untrusted Tcl scripts are prevented from corrupting the state of the hosting
application or computer. Untrusted scripts are also prevented from disclosing information stored on the
hosting computer or in the hosting application to any party.

The Safe Base allows a master interpreter to create safe, restricted interpreters that contain a set of predefined
aliases for the source, load, file, encoding, and exit commands and are able to use the auto–loading and
package mechanisms.

No knowledge of the file system structure is leaked to the safe interpreter, because it has access only to a
virtualized path containing tokens. When the safe interpreter requests to source a file, it uses the token in the
virtual path as part of the file name to source; the master interpreter transparently translates the token into a
real directory name and executes the requested operation (see the section SECURITY below for details). Different
levels of security can be selected by using the optional flags of the commands described below.

All commands provided in the master interpreter by the Safe Base reside in the safe namespace:

COMMANDS

The following commands are provided in the master interpreter:
::safe::interpCreate ?slave? ?options...?  
Creates a safe interpreter, installs the aliases described in the section ALIASES and initializes the auto-loading and package mechanism as specified by the supplied options. See the OPTIONS section below for a description of the optional arguments. If the slave argument is omitted, a name will be generated. ::safe::interpCreate always returns the interpreter name.

::safe::interpInit slave ?options...?  
This command is similar to interpCreate except it that does not create the safe interpreter. slave must have been created by some other means, like interp create −safe.

::safe::interpConfigure slave ?options...?  
If no options are given, returns the settings for all options for the named safe interpreter as a list of options and their current values for that slave. If a single additional argument is provided, it will return a list of 2 elements name and value where name is the full name of that option and value the current value for that option and the slave. If more than two additional arguments are provided, it will reconfigure the safe interpreter and change each and only the provided options. See the section on OPTIONS below for options description. Example of use:

```tcl
# Create a new interp with the same configuration as "$i0" :
set i1 [eval safe::interpCreate [safe::interpConfigure $i0]]
# Get the current deleteHook
set dh [safe::interpConfigure $i0 −del]
# Change (only) the statics loading ok attribute of an interp
# and its deleteHook (leaving the rest unchanged) :
safe::interpConfigure $i0 −delete {foo bar} −statics 0 ;
```

::safe::interpDelete slave  
Deletes the safe interpreter and cleans up the corresponding master interpreter data structures. If a deleteHook script was specified for this interpreter it is evaluated before the interpreter is deleted, with the name of the interpreter as an additional argument.

::safe::interpFindInAccessPath slave directory  
This command finds and returns the token for the real directory directory in the safe interpreter's current virtual access path. It generates an error if the directory is not found. Example of use:

```
$slave eval [list set tk_library [::safe::interpFindInAccessPath $name $tk_library]]
```

::safe::interpAddToAccessPath slave directory  
This command adds directory to the virtual path maintained for the safe interpreter in the master, and returns the token that can be used in the safe interpreter to obtain access to files in that directory. If the directory is already in the virtual path, it only returns the token without adding the directory to the virtual path again. Example of use:

```
$slave eval [list set tk_library [::safe::interpAddToAccessPath $name $tk_library]]
```

::safe::setLogCmd ?cmd arg...?  
This command installs a script that will be called when interesting life cycle events occur for a safe interpreter. When called with no arguments, it returns the currently installed script. When called with one argument, an empty string, the currently installed script is removed and logging is turned off. The script will be invoked with one additional argument, a string describing the event of interest. The main purpose is to help in debugging safe interpreters. Using this facility you can get complete error messages while the safe interpreter gets only generic error messages. This prevents a safe interpreter from seeing messages about failures and other events that might contain sensitive information such as
real directory names.
Example of use:

::safe::setLogCmd puts stderr

Below is the output of a sample session in which a safe interpreter attempted to source a file not found in its virtual access path. Note that the safe interpreter only received an error message saying that the file was not found:

NOTICE for slave interp10 : Created
NOTICE for slave interp10 : Setting accessPath=('/foo/bar') staticsok=1 nestedok=0 deletehook=()
NOTICE for slave interp10 : auto_path in interp10 has been set to {p(:0:)}
ERROR for slave interp10 : /foo/bar/init.tcl: no such file or directory

OPTIONS

The following options are common to ::safe::interpCreate, ::safe::interpInit, and ::safe::interpConfigure. Any option name can be abbreviated to its minimal non–ambiguous name. Option names are not case sensitive.

−accessPath directoryList
This option sets the list of directories from which the safe interpreter can source and load files. If this option is not specified, or if it is given as the empty list, the safe interpreter will use the same directories as its master for auto–loading. See the section SECURITY below for more detail about virtual paths, tokens and access control.

−statics boolean
This option specifies if the safe interpreter will be allowed to load statically linked packages (like load {} Tk). The default value is true : safe interpreters are allowed to load statically linked packages.

−noStatics
This option is a convenience shortcut for −statics false and thus specifies that the safe interpreter will not be allowed to load statically linked packages.

−nested boolean
This option specifies if the safe interpreter will be allowed to load packages into its own sub–interpreters. The default value is false : safe interpreters are not allowed to load packages into their own sub–interpreters.

−nestedLoadOk
This option is a convenience shortcut for −nested true and thus specifies the safe interpreter will be allowed to load packages into its own sub–interpreters.

−deleteHook script
When this option is given a non–empty script, it will be evaluated in the master with the name of the safe interpreter as an additional argument just before actually deleting the safe interpreter. Giving an empty value removes any currently installed deletion hook script for that safe interpreter. The default value ({}) is not to have any deletion call back.
ALIASES

The following aliases are provided in a safe interpreter:

source fileName
The requested file, a Tcl source file, is sourced into the safe interpreter if it is found. The source alias can only source files from directories in the virtual path for the safe interpreter. The source alias requires the safe interpreter to use one of the token names in its virtual path to denote the directory in which the file to be sourced can be found. See the section on SECURITY for more discussion of restrictions on valid filenames.

load fileName
The requested file, a shared object file, is dynamically loaded into the safe interpreter if it is found. The filename must contain a token name mentioned in the virtual path for the safe interpreter for it to be found successfully. Additionally, the shared object file must contain a safe entry point; see the manual page for the load command for more details.

file ?subCmd args...?
The file alias provides access to a safe subset of the subcommands of the file command; it allows only dirname, join, extension, root, tail, pathname and split subcommands. For more details on what these subcommands do see the manual page for the file command.

encoding ?subCmd args...?
The encoding alias provides access to a safe subset of the subcommands of the encoding command; it disallows setting of the system encoding, but allows all other subcommands including system to check the current encoding.

exit
The calling interpreter is deleted and its computation is stopped, but the Tcl process in which this interpreter exists is not terminated.

SECURITY

The Safe Base does not attempt to completely prevent annoyance and denial of service attacks. These forms of attack prevent the application or user from temporarily using the computer to perform useful work, for example by consuming all available CPU time or all available screen real estate. These attacks, while aggravating, are deemed to be of lesser importance in general than integrity and privacy attacks that the Safe Base is to prevent.

The commands available in a safe interpreter, in addition to the safe set as defined in interp manual page, are mediated aliases for source, load, exit, and safe subsets of file and encoding. The safe interpreter can also auto–load code and it can request that packages be loaded.

Because some of these commands access the local file system, there is a potential for information leakage about its directory structure. To prevent this, commands that take file names as arguments in a safe interpreter use tokens instead of the real directory names. These tokens are translated to the real directory name while a request to, e.g., source a file is mediated by the master interpreter. This virtual path system is maintained in the master interpreter for each safe interpreter created by ::safe::interpCreate or initialized by ::safe::interpInit and the path maps tokens accessible in the safe interpreter into real path names on the local
file system thus preventing safe interpreters from gaining knowledge about the structure of the file system of
the host on which the interpreter is executing. The only valid file names arguments for the source and load
aliases provided to the slave are path in the form of [file join token filename] (i.e. when using the native file
path formats: token\filename on Unix, token\filename on Windows, and token\filename on the Mac), where
token is representing one of the directories of the accessPath list and filename is one file in that directory (no
sub directories access are allowed).

When a token is used in a safe interpreter in a request to source or load a file, the token is checked and
translated to a real path name and the file to be sourced or loaded is located on the file system. The safe
interpreter never gains knowledge of the actual path name under which the file is stored on the file system.

To further prevent potential information leakage from sensitive files that are accidentally included in the set of
files that can be sourced by a safe interpreter, the source alias restricts access to files meeting the following
constraints: the file name must fourteen characters or shorter, must not contain more than one dot (","), must
end up with the extension .tcl or be called tclIndex.

Each element of the initial access path list will be assigned a token that will be set in the slave auto_path and
the first element of that list will be set as the tcl_library for that slave.

If the access path argument is not given or is the empty list, the default behavior is to let the slave access the
same packages as the master has access to (Or to be more precise: only packages written in Tcl (which by
definition can't be dangerous as they run in the slave interpreter) and C extensions that provides a Safe_Init
entry point). For that purpose, the master's auto_path will be used to construct the slave access path. In order
that the slave successfully loads the Tcl library files (which includes the auto−loading mechanism itself) the
tcl_library will be added or moved to the first position if necessary, in the slave access path, so the slave
tcl_library will be the same as the master's (its real path will still be invisible to the slave though). In order
that auto−loading works the same for the slave and the master in this by default case, the first−level sub
directories of each directory in the master auto_path will also be added (if not already included) to the slave
access path. You can always specify a more restrictive path for which sub directories will never be searched
by explicitly specifying your directory list with the −accessPath flag instead of relying on this default
mechanism.

When the accessPath is changed after the first creation or initialization (i.e. through interpConfigure
−accessPath list), an auto_reset is automatically evaluated in the safe interpreter to synchronize its
auto_index with the new token list.

SEE ALSO

interp, library, load, package, source, unknown

KEYWORDS

alias, auto−loading, auto_mkindex, load, master interpreter, safe interpreter, slave interpreter, source

Copyright ' 1995−1996 Sun Microsystems, Inc.
NAME

Tcl – Tool Command Language

SYNOPSIS

Summary of Tcl language syntax.

DESCRIPTION


$\text{name}$

$\text{name(index)}$

${\text{name}}$


\a
\b
\f
\n
\r
\t

\<newline>whiteSpace

\\\n\ooo
\xhh
\uhhhh


DESCRIPTION

The following rules define the syntax and semantics of the Tcl language:

[1] **Commands.**

A Tcl script is a string containing one or more commands. Semi−colons and newlines are command separators unless quoted as described below. Close brackets are command terminators during command substitution (see below) unless quoted.

[2] **Evaluation.**

A command is evaluated in two steps. First, the Tcl interpreter breaks the command into **words** and performs substitutions as described below. These substitutions are performed in the same way for all commands. The first word is used to locate a command procedure to carry out the command, then all of the words of the command are passed to the command procedure. The command procedure is free to interpret each of its words in any way it likes, such as an integer, variable name, list, or Tcl script. Different commands interpret their words differently.

[3] **Words.**

Words of a command are separated by white space (except for newlines, which are command separators).

[4] **Double quotes.**

If the first character of a word is double−quote (``''') then the word is terminated by the next double−quote character. If semi−colons, close brackets, or white space characters (including newlines) appear between the quotes then they are treated as ordinary characters and included in the word. Command substitution, variable substitution, and backslash substitution are performed on the characters between the quotes as described below. The double−quotes are not retained as part of the word.

[5] **Braces.**

If the first character of a word is an open brace (``{''') then the word is terminated by the matching close brace (``}'''). Braces nest within the word: for each additional open brace there must be an additional close brace (however, if an open brace or close brace within the word is quoted with a backslash then it is not counted in locating the matching close brace). No substitutions are performed on the characters between the braces except for backslash−newline substitutions described below, nor do semi−colons, newlines, close brackets, or white space receive any special interpretation. The word will consist of exactly the characters between the outer braces, not including the braces themselves.

[6] **Command substitution.**

If a word contains an open bracket (``[''') then Tcl performs **command substitution**. To do this it invokes the Tcl interpreter recursively to process the characters following the open bracket as a Tcl script. The script may contain any number of commands and must be terminated by a close bracket (``]'''). The result of the script (i.e. the result of its last command) is substituted into the word in place of the brackets and all of the characters between them. There may be any number of command substitutions in a single word. Command substitution is not performed on words enclosed in braces.

[7] **Variable substitution.**

If a word contains a dollar−sign (``$'') then Tcl performs **variable substitution**: the dollar−sign and the following characters are replaced in the word by the value of a variable. Variable substitution may take any of the following forms:
$\text{name}$

\textit{Name} is the name of a scalar variable; the name is a sequence of one or more characters that are a letter, digit, underscore, or namespace separators (two or more colons).

$\text{name(index)}$

\textit{Name} gives the name of an array variable and \textit{index} gives the name of an element within that array. \textit{Name} must contain only letters, digits, underscores, and namespace separators, and may be an empty string. Command substitutions, variable substitutions, and backslash substitutions are performed on the characters of \textit{index}.

$\{\text{name}\}$

\textit{Name} is the name of a scalar variable. It may contain any characters whatsoever except for close braces.

There may be any number of variable substitutions in a single word. Variable substitution is not performed on words enclosed in braces.

\textbf{[8] Backslash substitution.}

If a backslash (``\'') appears within a word then \textit{backslash substitution} occurs. In all cases but those described below the backslash is dropped and the following character is treated as an ordinary character and included in the word. This allows characters such as double quotes, close brackets, and dollar signs to be included in words without triggering special processing. The following table lists the backslash sequences that are handled specially, along with the value that replaces each sequence.

\begin{itemize}
  \item \texttt{\textbackslash a} \quad \text{Audible alert (bell) (0x7).}
  \item \texttt{\textbackslash b} \quad \text{Backspace (0x8).}
  \item \texttt{\textbackslash f} \quad \text{Form feed (0xc).}
  \item \texttt{\textbackslash n} \quad \text{Newline (0xa).}
  \item \texttt{\textbackslash r} \quad \text{Carriage−return (0xd).}
  \item \texttt{\textbackslash t} \quad \text{Tab (0x9).}
  \item \texttt{\textbackslash v} \quad \text{Vertical tab (0xb).}
  \item \texttt{\textbackslash \}<newline>\textit{whiteSpace} \quad \text{A single space character replaces the backslash, newline, and all spaces and tabs after the newline. This backslash sequence is unique in that it is replaced in a separate pre−pass before the command is actually parsed. This means that it will be replaced even when it occurs between braces, and the resulting space will be treated as a word separator if it isn't in braces or quotes.}
  \item \texttt{\textbackslash \textbackslash} \quad \text{Backslash (``\textbackslash\ '').}
  \item \texttt{\textbackslash \textsc{ooo}} \quad \text{The digits ooo (one, two, or three of them) give an eight−bit octal value for the Unicode character that will be inserted. The upper bits of the Unicode character will be 0.}
\end{itemize}
\xhh
The hexadecimal digits \hh give an eight−bit hexadecimal value for the Unicode character that will be inserted. Any number of hexadecimal digits may be present; however, all but the last two are ignored (the result is always a one−byte quantity). The upper bits of the Unicode character will be 0.

\uhhhh
The hexadecimal digits \hhhh (one, two, three, or four of them) give a sixteen−bit hexadecimal value for the Unicode character that will be inserted.

Backslash substitution is not performed on words enclosed in braces, except for backslash−newline as described above.

If a hash character (``#``) appears at a point where Tcl is expecting the first character of the first word of a command, then the hash character and the characters that follow it, up through the next newline, are treated as a comment and ignored. The comment character only has significance when it appears at the beginning of a command.

Each character is processed exactly once by the Tcl interpreter as part of creating the words of a command. For example, if variable substitution occurs then no further substitutions are performed on the value of the variable; the value is inserted into the word verbatim. If command substitution occurs then the nested command is processed entirely by the recursive call to the Tcl interpreter; no substitutions are performed before making the recursive call and no additional substitutions are performed on the result of the nested script.

Substitutions take place from left to right, and each substitution is evaluated completely before attempting to evaluate the next. Thus, a sequence like

```
set y [set x 0][incr x][incr x]
```

will always set the variable \y to the value, 012.

Substitutions do not affect the word boundaries of a command. For example, during variable substitution the entire value of the variable becomes part of a single word, even if the variable's value contains spaces.
NAME

after – Execute a command after a time delay

SYNOPSIS

after ms
after ms ?script script script ...?
after cancel id
after cancel script script script ...
after idle ?script script script ...
after info ?id?

DESCRIPTION

This command is used to delay execution of the program or to execute a command in background sometime in the future. It has several forms, depending on the first argument to the command:

after ms
Ms must be an integer giving a time in milliseconds. The command sleeps for ms milliseconds and then returns. While the command is sleeping the application does not respond to events.

**after ms ?script script script ...?**

In this form the command returns immediately, but it arranges for a Tcl command to be executed ms milliseconds later as an event handler. The command will be executed exactly once, at the given time. The delayed command is formed by concatenating all the script arguments in the same fashion as the **concat** command. The command will be executed at global level (outside the context of any Tcl procedure). If an error occurs while executing the delayed command then the **bgerror** mechanism is used to report the error. The **after** command returns an identifier that can be used to cancel the delayed command using **after cancel**.

**after cancel id**

Cancels the execution of a delayed command that was previously scheduled. Id indicates which command should be canceled; it must have been the return value from a previous after command. If the command given by id has already been executed then the **after cancel** command has no effect.

**after cancel script script ...**

This command also cancels the execution of a delayed command. The script arguments are concatenated together with space separators (just as in the **concat** command). If there is a pending command that matches the string, it is cancelled and will never be executed; if no such command is currently pending then the **after cancel** command has no effect.

**after idle script ?script script ...?**

Concatenates the script arguments together with space separators (just as in the **concat** command), and arranges for the resulting script to be evaluated later as an idle callback. The script will be run exactly once, the next time the event loop is entered and there are no events to process. The command returns an identifier that can be used to cancel the delayed command using **after cancel**. If an error occurs while executing the script then the **bgerror** mechanism is used to report the error.

**after info ?id?**

This command returns information about existing event handlers. If no id argument is supplied, the command returns a list of the identifiers for all existing event handlers created by the **after** command for this interpreter. If id is supplied, it specifies an existing handler; id must have been the return value from some previous call to **after** and it must not have triggered yet or been cancelled. In this case the command returns a list with two elements. The first element of the list is the script associated with id, and the second element is either idle or timer to indicate what kind of event handler it is.

The **after ms** and **after idle** forms of the command assume that the application is event driven: the delayed commands will not be executed unless the application enters the event loop. In applications that are not normally event–driven, such as **tclsh**, the event loop can be entered with the **vwait** and **update** commands.

### EXAMPLES

This defines a command to make Tcl do nothing at all for N seconds:

```tcl
proc sleep {N} {
    after [expr {int($N * 1000)}]
}
```
This arranges for the command `wake_up` to be run in eight hours (providing the event loop is active at that time):

```
after [expr {1000 * 60 * 60 * 8}] wake_up
```

The following command can be used to do long-running calculations (as represented here by `::my_calc::one_step`, which is assumed to return a boolean indicating whether another step should be performed) in a step-by-step fashion, though the calculation itself needs to be arranged so it can work step-wise. This technique is extra careful to ensure that the event loop is not starved by the rescheduling of processing steps (arranging for the next step to be done using an already-triggered timer event only when the event queue has been drained) and is useful when you want to ensure that a Tk GUI remains responsive during a slow task.

```
proc doOneStep {} {
    if {[:my_calc::one_step]} {
        after idle [list after 0 doOneStep]
    }
}
doOneStep
```

SEE ALSO

`bgerror`, `concat`, `update`, `vwait`

KEYWORDS

`cancel`, `delay`, `idle callback`, `sleep`, `time`

Copyright © 1990–1994 The Regents of the University of California.
Copyright © 1994–1996 Sun Microsystems, Inc.
append

NAME

append – Append to variable

SYNOPSIS

append varName ?value value value ...?

DESCRIPTION

Append all of the value arguments to the current value of variable varName. If varName doesn’t exist, it is given a value equal to the concatenation of all the value arguments. The result of this command is the new value stored in variable varName. This command provides an efficient way to build up long variables incrementally. For example, “append a $b” is much more efficient than “set a $a$b” if $a$ is long.

EXAMPLE

Building a string of comma-separated numbers piecemeal using a loop.

    set var 0
    for {set i 1} {$i<=10} {incr i} {
        append var "," $i
    }
    puts $var
    # Prints 0,1,2,3,4,5,6,7,8,9,10

SEE ALSO

concat, lappend

KEYWORDS

append, variable

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

array – Manipulate array variables

SYNOPSIS

array option arrayName ?arg arg ...?

DESCRIPTION

array anymore arrayName searchId
array donesearch arrayName searchId
array exists arrayName
array get arrayName ?pattern?
array names arrayName ?mode? ?pattern?
array nextelement arrayName searchId
array set arrayName list
array size arrayName
array startsearch arrayName
array statistics arrayName
array unset arrayName ?pattern?

EXAMPLES

SEE ALSO

KEYWORDS
indicate whether the search has been completed.

**array donesearch arrayName searchId**

This command terminates an array search and destroys all the state associated with that search. *SearchId* indicates which search on *arrayName* to destroy, and must have been the return value from a previous invocation of **array startsearch**. Returns an empty string.

**array exists arrayName**

Returns 1 if *arrayName* is an array variable, 0 if there is no variable by that name or if it is a scalar variable.

**array get arrayName ?pattern?**

Returns a list containing pairs of elements. The first element in each pair is the name of an element in *arrayName* and the second element of each pair is the value of the array element. The order of the pairs is undefined. If *pattern* is not specified, then all of the elements of the array are included in the result. If *pattern* is specified, then only those elements whose names match *pattern* (using the matching rules of **string match**) are included. If *arrayName* isn't the name of an array variable, or if the array contains no elements, then an empty list is returned.

**array names arrayName ?mode? ?pattern?**

Returns a list containing the names of all of the elements in the array that match *pattern*. *Mode* may be one of −exact, −glob, or −regexp. If specified, *mode* designates which matching rules to use to match *pattern* against the names of the elements in the array. If not specified, *mode* defaults to −glob. See the documentation for **string match** for information on glob style matching, and the documentation for **regexp** for information on regexp matching. If *pattern* is omitted then the command returns all of the element names in the array. If there are no (matching) elements in the array, or if *arrayName* isn't the name of an array variable, then an empty string is returned.

**array nextelement arrayName searchId**

Returns the name of the next element in *arrayName*, or an empty string if all elements of *arrayName* have already been returned in this search. The *searchId* argument identifies the search, and must have been the return value of an **array startsearch** command. Warning: if elements are added to or deleted from the array, then all searches are automatically terminated just as if **array donesearch** had been invoked; this will cause **array nextelement** operations to fail for those searches.

**array set arrayName list**

Sets the values of one or more elements in *arrayName*. *list* must have a form like that returned by **array get**, consisting of an even number of elements. Each odd–numbered element in *list* is treated as an element name within *arrayName*, and the following element in *list* is used as a new value for that array element. If the variable *arrayName* does not already exist and *list* is empty, *arrayName* is created with an empty array value.

**array size arrayName**

Returns a decimal string giving the number of elements in the array. If *arrayName* isn't the name of an array then 0 is returned.

**array startsearch arrayName**

This command initializes an element–by–element search through the array given by *arrayName*, such that invocations of the **array nextelement** command will return the names of the individual elements in the array. When the search has been completed, the **array donesearch** command should be invoked. The return value is a search identifier that must be used in **array nextelement** and **array donesearch** commands; it allows multiple searches to be underway simultaneously for the same array. It is currently more efficient and easier to use either the **array get** or **array names**, together
with `foreach`, to iterate over all but very large arrays. See the examples below for how to do this.

**array statistics** `arrayName`

Returns statistics about the distribution of data within the hashtable that represents the array. This information includes the number of entries in the table, the number of buckets, and the utilization of the buckets.

**array unset** `arrayName` `?pattern??`

Unsets all of the elements in the array that match `pattern` (using the matching rules of `string match`). If `arrayName` isn't the name of an array variable or there are no matching elements in the array, no error will be raised. If `pattern` is omitted and `arrayName` is an array variable, then the command unsets the entire array. The command always returns an empty string.

### EXAMPLES

```tcl
array set colorcount {
    red   1
    green 5
    blue  4
    white 9
}

foreach {color count} [array get colorcount] {
    puts "Color: $color Count: $count"
}
=> Color: blue Count: 4
    Color: white Count: 9
    Color: green Count: 5
    Color: red Count: 1

foreach color [array names colorcount] {
    puts "Color: $color Count: [array count colorcount($color)]"
}
=> Color: blue Count: 4
    Color: white Count: 9
    Color: green Count: 5
    Color: red Count: 1

foreach color [lsort [array names colorcount]] {
    puts "Color: $color Count: [array count colorcount($color)]"
}
=> Color: blue Count: 4
    Color: green Count: 5
    Color: red Count: 1
    Color: white Count: 9

array statistics colorcount
=> 4 entries in table, 4 buckets
    number of buckets with 0 entries: 1
    number of buckets with 1 entries: 2
    number of buckets with 2 entries: 1
    number of buckets with 3 entries: 0
    number of buckets with 4 entries: 0
    number of buckets with 5 entries: 0
```
number of buckets with 6 entries: 0
number of buckets with 7 entries: 0
number of buckets with 8 entries: 0
number of buckets with 9 entries: 0
number of buckets with 10 or more entries: 0
average search distance for entry: 1.2

SEE ALSO

list, string, variable, trace, foreach

KEYWORDS

array, element names, search

Copyright ’ 1993−1994 The Regents of the University of California.
Copyright ’ 1994−1996 Sun Microsystems, Inc.
Copyright ’ 1995−1997 Roger E. Critchlow Jr.
bgerror

NAME

bgerror – Command invoked to process background errors

SYNOPSIS

bgerror message

DESCRIPTION

The bgerror command doesn't exist as built-in part of Tcl. Instead, individual applications or users can define a bgerror command (e.g. as a Tcl procedure) if they wish to handle background errors.

A background error is one that occurs in an event handler or some other command that didn't originate with the application. For example, if an error occurs while executing a command specified with the after command, then it is a background error. For a non-background error, the error can simply be returned up through nested Tcl command evaluations until it reaches the top-level code in the application; then the application can report the error in whatever way it wishes. When a background error occurs, the unwinding ends in the Tcl library and there is no obvious way for Tcl to report the error.

When Tcl detects a background error, it saves information about the error and invokes the bgerror command later as an idle event handler. Before invoking bgerror, Tcl restores the errorInfo and errorCode variables to their values at the time the error occurred, then it invokes bgerror with the error message as its only argument. Tcl assumes that the application has implemented the bgerror command, and that the command will report the error in a way that makes sense for the application. Tcl will ignore any result returned by the bgerror command as long as no error is generated.

If another Tcl error occurs within the bgerror command (for example, because no bgerror command has been defined) then Tcl reports the error itself by writing a message to stderr.

If several background errors accumulate before bgerror is invoked to process them, bgerror will be invoked once for each error, in the order they occurred. However, if bgerror returns with a break exception, then any remaining errors are skipped without calling bgerror.

Tcl has no default implementation for bgerror. However, in applications using Tk there is a default bgerror procedure which posts a dialog box containing the error message and offers the user a chance to see a stack trace showing where the error occurred. In addition to allowing the user to view the stack trace, the dialog provides an additional application configurable button which may be used, for example, to save the stack trace to a file. By default, this is the behavior associated with that button. This behavior can be redefined by setting the option database values *ErrorDialog.function.text, to specify the caption for the function button, and *ErrorDialog.function.command, to specify the command to be run. The text of the stack trace is appended to the command when it is evaluated. If either of these options is set to the empty string, then the additional
button will not be displayed in the dialog.

If you are writing code that will be used by others as part of a package or other kind of library, consider avoiding `bgerror`. The reason for this is that the application programmer may also want to define a `bgerror`, or use other code that does and thus will have trouble integrating your code.

**EXAMPLE**

This `bgerror` procedure appends errors to a file, with a timestamp.

```tcl
proc bgerror {message} {
    set timestamp [clock format [clock seconds]]
    set fl [open mylog.txt {WRONLY CREAT APPEND}]
    puts $fl "$timestamp: bgerror in $::argv "$message""
    close $fl
}
```

**SEE ALSO**

`after`, `tclvars`

**KEYWORDS**

`background error, reporting`

---

*Copyright ' 1990−1994 The Regents of the University of California.  
Copyright ' 1994−1996 Sun Microsystems, Inc.  
Copyright ' 1995−1997 Roger E. Critchlow Jr.*
binary

NAME

binary – Insert and extract fields from binary strings

SYNOPSIS

binary format formatString ?arg arg ...?

binary scan string formatString ?varName varName ...?

DESCRIPTION

BINARY FORMAT

a
A
b
B
h
H

BINARY SCAN

a
A
b
B
NAME

binary – Insert and extract fields from binary strings

SYNOPSIS

binary format formatString ?arg arg ...?
binary scan string formatString ?varName varName ...?

DESCRIPTION

This command provides facilities for manipulating binary data. The first form, binary format, creates a binary string from normal Tcl values. For example, given the values 16 and 22, on a 32 bit architecture, it might produce an 8-byte binary string consisting of two 4-byte integers, one for each of the numbers. The second form of the command, binary scan, does the opposite: it extracts data from a binary string and returns it as ordinary Tcl string values.

BINARY FORMAT

The binary format command generates a binary string whose layout is specified by the formatString and whose contents come from the additional arguments. The resulting binary value is returned.

The formatString consists of a sequence of zero or more field specifiers separated by zero or more spaces. Each field specifier is a single type character followed by an optional numeric count. Most field specifiers consume one argument to obtain the value to be formatted. The type character specifies how the value is to be formatted. The count typically indicates how many items of the specified type are taken from the value. If present, the count is a non-negative decimal integer or *, which normally indicates that all of the items in the value are to be used. If the number of arguments does not match the number of fields in the format string that consume arguments, then an error is generated.

Here is a small example to clarify the relation between the field specifiers and the arguments:

binary format d3d {1.0 2.0 3.0 4.0} 0.1

The first argument is a list of four numbers, but because of the count of 3 for the associated field specifier, only the first three will be used. The second argument is associated with the second field specifier. The resulting binary string contains the four numbers 1.0, 2.0, 3.0 and 0.1.
Each type–count pair moves an imaginary cursor through the binary data, storing bytes at the current position and advancing the cursor to just after the last byte stored. The cursor is initially at position 0 at the beginning of the data. The type may be any one of the following characters:

a
Stores a character string of length count in the output string. Every character is taken as modulo 256 (i.e. the low byte of every character is used, and the high byte discarded) so when storing character strings not wholly expressible using the characters \u0000–\u00ff, the encoding convertto command should be used first if this truncation is not desired (i.e. if the characters are not part of the ISO 8859–1 character set.) If arg has fewer than count bytes, then additional zero bytes are used to pad out the field. If arg is longer than the specified length, the extra characters will be ignored. If count is *, then all of the bytes in arg will be formatted. If count is omitted, then one character will be formatted. For example,

binary format a7a*a alpha bravo charlie

will return a string equivalent to alpha\000\000bravoc.

A
This form is the same as a except that spaces are used for padding instead of nulls. For example,

binary format A6A*A alpha bravo charlie

will return alpha bravoc.

b
Stores a string of count binary digits in low–to–high order within each byte in the output string. Arg must contain a sequence of 1 and 0 characters. The resulting bytes are emitted in first to last order with the bits being formatted in low–to–high order within each byte. If arg has fewer than count digits, then zeros will be used for the remaining bits. If arg has more than the specified number of digits, the extra digits will be ignored. If count is *, then all of the digits in arg will be formatted. If count is omitted, then one digit will be formatted. If the number of bits formatted does not end at a byte boundary, the remaining bits of the last byte will be zeros. For example,

binary format b5b* 11100 111000011010

will return a string equivalent to \x07\x87\x05.

B
This form is the same as b except that the bits are stored in high–to–low order within each byte. For example,

binary format B5B* 11100 111000011010

will return a string equivalent to \xe0\xe1\xa0.

h
Stores a string of count hexadecimal digits in low–to–high within each byte in the output string. Arg must contain a sequence of characters in the set ```0123456789abcdefABCDEF```. The resulting bytes are emitted in first to last order with the hex digits being formatted in low–to–high order within each
byte. If `arg` has fewer than `count` digits, then zeros will be used for the remaining digits. If `arg` has more than the specified number of digits, the extra digits will be ignored. If `count` is *, then all of the digits in `arg` will be formatted. If `count` is omitted, then one digit will be formatted. If the number of digits formatted does not end at a byte boundary, the remaining bits of the last byte will be zeros. For example,

```
binary format h3h* AB def
```

will return a string equivalent to `\xba\x00\xed\x0f`.

**H**

This form is the same as `h` except that the digits are stored in high−to−low order within each byte. For example,

```
binary format H3H* ab DEF
```

will return a string equivalent to `\xab\x00\xde\xf0`.

**c**

Stores one or more 8−bit integer values in the output string. If no `count` is specified, then `arg` must consist of an integer value; otherwise `arg` must consist of a list containing at least `count` integer elements. The low−order 8 bits of each integer are stored as a one−byte value at the cursor position. If `count` is *, then all of the integers in the list are formatted. If the number of elements in the list is fewer than `count`, then an error is generated. If the number of elements in the list is greater than `count`, then the extra elements are ignored. For example,

```
binary format c3cc* {3 −3 128 1} 260 {2 5}
```

will return a string equivalent to `\x03\xfd\x80\x04\x02\x05`, whereas

```
binary format c {2 5}
```

will generate an error.

**s**

This form is the same as `c` except that it stores one or more 16−bit integers in little−endian byte order in the output string. The low−order 16−bits of each integer are stored as a two−byte value at the cursor position with the least significant byte stored first. For example,

```
binary format s3 {3 −3 258 1}
```

will return a string equivalent to `\x03\x00\xfd\xff\x02\x01`.

**S**

This form is the same as `s` except that it stores one or more 16−bit integers in big−endian byte order in the output string. For example,

```
binary format S3 {3 −3 258 1}
```

will return a string equivalent to `\x00\x03\xff\xfd\x01\x02`. 
This form is the same as \texttt{c} except that it stores one or more 32–bit integers in little–endian byte order in the output string. The low–order 32–bits of each integer are stored as a four–byte value at the cursor position with the least significant byte stored first. For example,

\begin{verbatim}
    binary format i3 {3 \ -3 \ 65536 \ 1}
\end{verbatim}

will return a string equivalent to \verb|x03|x00|x00|x00|x00|xfd|xff|xff|xff|x00|x00|x00|x01|x00|

\textbf{I}

This form is the same as \texttt{i} except that it stores one or more one or more 32–bit integers in big–endian byte order in the output string. For example,

\begin{verbatim}
    binary format I3 {3 \ -3 \ 65536 \ 1}
\end{verbatim}

will return a string equivalent to \verb|x00|x00|x00|x00|x00|x00|x00|x00|x00|x00|x00|x00|x00|x00|

\textbf{W}

This form is the same as \texttt{c} except that it stores one or more 64–bit integers in little–endian byte order in the output string. The low–order 64–bits of each integer are stored as an eight–byte value at the cursor position with the least significant byte stored first. For example,

\begin{verbatim}
    binary format w 7810179016327718216
\end{verbatim}

will return the string \texttt{HelloTcl}

\textbf{W}

This form is the same as \texttt{w} except that it stores one or more one or more 64–bit integers in big–endian byte order in the output string. For example,

\begin{verbatim}
    binary format Wc 4785469626960341345 \ 110
\end{verbatim}

will return the string \texttt{BigEndian}

\textbf{f}

This form is the same as \texttt{c} except that it stores one or more one or more single–precision floating in the machine's native representation in the output string. This representation is not portable across architectures, so it should not be used to communicate floating point numbers across the network. The size of a floating point number may vary across architectures, so the number of bytes that are generated may vary. If the value overflows the machine's native representation, then the value of FLT\_MAX as defined by the system will be used instead. Because Tcl uses double–precision floating–point numbers internally, there may be some loss of precision in the conversion to single–precision. For example, on a Windows system running on an Intel Pentium processor,

\begin{verbatim}
    binary format f2 {1.6 \ 3.4}
\end{verbatim}

will return a string equivalent to \verb|xcd|\verb|xcc|\verb|xcc|\verb|x3f|\verb|x9a|\verb|x99|\verb|x59|\verb|x40|.

\textbf{d}

This form is the same as \texttt{f} except that it stores one or more one or more double–precision floating in the machine's native representation in the output string. For example, on a Windows system running
on an Intel Pentium processor,

\texttt{binary format d1 \{1.6\}}

will return a string equivalent to \texttt{x9a\x99\x99\x99\x99\x99\x9f3f}.

\text{x}
Stores \textit{count} null bytes in the output string. If \textit{count} is not specified, stores one null byte. If \textit{count} is *, generates an error. This type does not consume an argument. For example,

\texttt{binary format a3xa3x2a3 abc def ghi}

will return a string equivalent to \texttt{abc\000def\000\000ghi}.

\text{x}
Moves the cursor back \textit{count} bytes in the output string. If \textit{count} is * or is larger than the current cursor position, then the cursor is positioned at location 0 so that the next byte stored will be the first byte in the result string. If \textit{count} is omitted then the cursor is moved back one byte. This type does not consume an argument. For example,

\texttt{binary format a3X*a3X2a3 abc def ghi}

will return \texttt{dghi}.

\text@}
Moves the cursor to the absolute location in the output string specified by \textit{count}. Position 0 refers to the first byte in the output string. If \textit{count} refers to a position beyond the last byte stored so far, then null bytes will be placed in the uninitialized locations and the cursor will be placed at the specified location. If \textit{count} is *, then the cursor is moved to the current end of the output string. If \textit{count} is omitted, then an error will be generated. This type does not consume an argument. For example,

\texttt{binary format a5@2a1@*a3@10a1 abcde f ghi j}

will return \texttt{abfdeghi\000\000j}.

**BINARY SCAN**

The \texttt{binary scan} command parses fields from a binary string, returning the number of conversions performed. \texttt{String} gives the input to be parsed and \texttt{formatString} indicates how to parse it. Each \texttt{varName} gives the name of a variable; when a field is scanned from \texttt{string} the result is assigned to the corresponding variable.

As with \texttt{binary format}, the \texttt{formatString} consists of a sequence of zero or more field specifiers separated by zero or more spaces. Each field specifier is a single type character followed by an optional numeric \textit{count}. Most field specifiers consume one argument to obtain the variable into which the scanned values should be placed. The type character specifies how the binary data is to be interpreted. The \textit{count} typically indicates how many items of the specified type are taken from the data. If present, the \textit{count} is a non-negative decimal integer or *, which normally indicates that all of the remaining items in the data are to be used. If there are not enough bytes left after the current cursor position to satisfy the current field specifier, then the corresponding variable is left untouched and \texttt{binary scan} returns immediately with the number of variables that were set. If
there are not enough arguments for all of the fields in the format string that consume arguments, then an error is generated.

A similar example as with binary format should explain the relation between field specifiers and arguments in case of the binary scan subcommand:

```
binary scan $bytes s3s first second
```

This command (provided the binary string in the variable bytes is long enough) assigns a list of three integers to the variable first and assigns a single value to the variable second. If bytes contains fewer than 8 bytes (i.e. four 2–byte integers), no assignment to second will be made, and if bytes contains fewer than 6 bytes (i.e. three 2–byte integers), no assignment to first will be made. Hence:

```
puts [binary scan abcdefg s3s first second]
puts $first
puts $second
```

will print (assuming neither variable is set previously):

```
1
25185 25699 26213
can't read "second": no such variable
```

It is important to note that the c, s, and S (and i and I on 64bit systems) will be scanned into long data size values. In doing this, values that have their high bit set (0x80 for chars, 0x8000 for shorts, 0x80000000 for ints), will be sign extended. Thus the following will occur:

```
set signShort [binary format s1 0x8000]
binary scan $signShort s1 val; # val == 0xFFFF8000
```

If you want to produce an unsigned value, then you can mask the return value to the desired size. For example, to produce an unsigned short value:

```
set val [expr {$val & 0xFFFF}]; # val == 0x8000
```

Each type–count pair moves an imaginary cursor through the binary data, reading bytes from the current position. The cursor is initially at position 0 at the beginning of the data. The type may be any one of the following characters:

\textit{a}

The data is a character string of length \textit{count}. If \textit{count} is *, then all of the remaining bytes in \textit{string} will be scanned into the variable. If \textit{count} is omitted, then one character will be scanned. All characters scanned will be interpreted as being in the range \texttt{\textbackslash u0000–\textbackslash u00ff} so the \texttt{encoding convertfrom} command might be needed if the string is not an ISO 8859–1 string. For example,

```
binary scan abcde\000fghi a6a10 var1 var2
```
A

This form is the same as a, except trailing blanks and nulls are stripped from the scanned value before it is stored in the variable. For example,

```
binary scan "abc efghi \000" A* var1
```

will return 1 with abc efghi stored in var1.

b

The data is turned into a string of count binary digits in low–to–high order represented as a sequence of "1" and "0" characters. The data bytes are scanned in first to last order with the bits being taken in low–to–high order within each byte. Any extra bits in the last byte are ignored. If count is *, then all of the remaining bits in string will be scanned. If count is omitted, then one bit will be scanned. For example,

```
binary scan \x07\x87\x05 b5b* var1 var2
```

will return 2 with 11100 stored in var1 and 1110000110100000 stored in var2.

B

This form is the same as b, except the bits are taken in high–to–low order within each byte. For example,

```
binary scan \x70\x87\x05 B5B* var1 var2
```

will return 2 with 01110 stored in var1 and 1000011100000101 stored in var2.

h

The data is turned into a string of count hexadecimal digits in low–to–high order represented as a sequence of characters in the set "0123456789abcdef". The data bytes are scanned in first to last order with the hex digits being taken in low–to–high order within each byte. Any extra bits in the last byte are ignored. If count is *, then all of the remaining hex digits in string will be scanned. If count is omitted, then one hex digit will be scanned. For example,

```
binary scan \x07\x86\x05 h3h* var1 var2
```

will return 2 with 706 stored in var1 and 50 stored in var2.

H

This form is the same as h, except the digits are taken in high–to–low order within each byte. For example,

```
binary scan \x07\x86\x05 H3H* var1 var2
```

will return 2 with 078 stored in var1 and 05 stored in var2.

c

The data is turned into count 8–bit signed integers and stored in the corresponding variable as a list. If count is *, then all of the remaining bytes in string will be scanned. If count is omitted, then one 8–bit integer will be scanned. For example,
binary scan \x07\x86\x05 c2c* var1 var2

will return 2 with 7 –122 stored in var1 and 5 stored in var2. Note that the integers returned are signed, but they can be converted to unsigned 8–bit quantities using an expression like:

```
expr ( $num + 0x100 ) % 0x100
```

The data is interpreted as count 16–bit signed integers represented in little–endian byte order. The integers are stored in the corresponding variable as a list. If count is *, then all of the remaining bytes in string will be scanned. If count is omitted, then one 16–bit integer will be scanned. For example,

```
binary scan \x05\x00\x07\x00\xf0\xff s2s* var1 var2
```

will return 2 with 5 7 stored in var1 and −16 stored in var2. Note that the integers returned are signed, but they can be converted to unsigned 16–bit quantities using an expression like:

```
expr ( $num + 0x10000 ) % 0x10000
```

This form is the same as s except that the data is interpreted as count 16–bit signed integers represented in big–endian byte order. For example,

```
binary scan \x00\x05\x00\x07\xff\xf0 S2S* var1 var2
```

will return 2 with 5 7 stored in var1 and −16 stored in var2.

i

The data is interpreted as count 32–bit signed integers represented in little–endian byte order. The integers are stored in the corresponding variable as a list. If count is *, then all of the remaining bytes in string will be scanned. If count is omitted, then one 32–bit integer will be scanned. For example,

```
binary scan \x05\x00\x00\x00\x07\x00\x00\x00\xf0\xff\xff\xff i2i* var1 var2
```

will return 2 with 5 7 stored in var1 and −16 stored in var2. Note that the integers returned are signed and cannot be represented by Tcl as unsigned values.

I

This form is the same as I except that the data is interpreted as count 32–bit signed integers represented in big–endian byte order. For example,

```
binary scan \x00\x00\x00\x05\x00\x00\x00\x00\x07\xff\xff\xff\xff I2I* var1 var2
```

will return 2 with 5 7 stored in var1 and −16 stored in var2.

w

The data is interpreted as count 64–bit signed integers represented in little–endian byte order. The integers are stored in the corresponding variable as a list. If count is *, then all of the remaining bytes in string will be scanned. If count is omitted, then one 64–bit integer will be scanned. For example,

```
binary scan \x05\x00\x00\x00\x07\x00\x00\x00\x00\xf0\xff\xff\xff\xff wi* var1 var2
```
will return 2 with 30064771077 stored in var1 and −16 stored in var2. Note that the integers returned are signed and cannot be represented by Tcl as unsigned values.

W

This form is the same as w except that the data is interpreted as count 64−bit signed integers represented in big−endian byte order. For example,

```
binary scan \x00\x00\x00\x05\x00\x00\x00\x07\xff\xff\xff\xff\xff\xff\xff\x0f WI* var1 var2
```

will return 2 with 21474836487 stored in var1 and −16 stored in var2.

f

The data is interpreted as count single−precision floating point numbers in the machine’s native representation. The floating point numbers are stored in the corresponding variable as a list. If count is *, then all of the remaining bytes in \texttt{string} will be scanned. If count is omitted, then one single−precision floating point number will be scanned. The size of a floating point number may vary across architectures, so the number of bytes that are scanned may vary. If the data does not represent a valid floating point number, the resulting value is undefined and compiler dependent. For example, on a Windows system running on an Intel Pentium processor,

```
binary scan \x3f\xcc\xcc\xcd f var1
```

will return 1 with 1.6000000238418579 stored in var1.

d

This form is the same as f except that the data is interpreted as count double−precision floating point numbers in the machine’s native representation. For example, on a Windows system running on an Intel Pentium processor,

```
binary scan \x9a\x99\x99\x99\x99\x99\xf9\x3f d var1
```

will return 1 with 1.6000000000000001 stored in var1.

x

Moves the cursor forward count bytes in \texttt{string}. If count is * or is larger than the number of bytes after the current cursor cursor position, then the cursor is positioned after the last byte in \texttt{string}. If count is omitted, then the cursor is moved forward one byte. Note that this type does not consume an argument. For example,

```
binary scan \x01\x02\x03\x04 x2H* var1
```

will return 1 with 0304 stored in var1.

X

Moves the cursor back count bytes in \texttt{string}. If count is * or is larger than the current cursor position, then the cursor is positioned at location 0 so that the next byte scanned will be the first byte in \texttt{string}. If count is omitted then the cursor is moved back one byte. Note that this type does not consume an argument. For example,

```
binary scan \x01\x02\x03\x04 c2X* var1 var2
```
will return 2 with 1 2 stored in var1 and 020304 stored in var2.

@
Moves the cursor to the absolute location in the data string specified by count. Note that position 0 refers to the first byte in string. If count refers to a position beyond the end of string, then the cursor is positioned after the last byte. If count is omitted, then an error will be generated. For example,

binary scan \x01\x02\x03\x04 c2@1H* var1 var2

will return 2 with 1 2 stored in var1 and 020304 stored in var2.

PLATFORM ISSUES

Sometimes it is desirable to format or scan integer values in the native byte order for the machine. Refer to the byteOrder element of the tcl_platform array to decide which type character to use when formatting or scanning integers.

EXAMPLES

This is a procedure to write a Tcl string to a binary-encoded channel as UTF-8 data preceded by a length word:

```tcl
proc writeString {channel string} {
    set data [encoding convertto utf-8 $string]
    puts −nonewline [binary format Ia* \[string length $data\] $data]
}
```

This procedure reads a string from a channel that was written by the previously presented `writeString` procedure:

```tcl
proc readString {channel} {
    if {![binary scan [read $channel 4] I length]} {
        error "missing length"
    } else {
        set data [read $channel $length]
        return [encoding convertfrom utf-8 $data]
    }
}
```

SEE ALSO

format, scan, tclvars

KEYWORDS

binary, format, scan

Copyright ' 1997 by Sun Microsystems, Inc.
break

NAME

break – Abort looping command

SYNOPSIS

break

DESCRIPTION

This command is typically invoked inside the body of a looping command such as for or foreach or while. It returns a TCL_BREAK code, which causes a break exception to occur. The exception causes the current script to be aborted out to the innermost containing loop command, which then aborts its execution and returns normally. Break exceptions are also handled in a few other situations, such as the catch command, Tk event bindings, and the outermost scripts of procedure bodies.

EXAMPLE

Print a line for each of the integers from 0 to 5:

```
for {set x 0} {$x<10} {incr x} {
    if {$x > 5} {
        break
    } else {
        puts "x is $x"
    }
}
```

SEE ALSO

catch, continue, for, foreach, return, while

KEYWORDS

abort, break, loop

Copyright ' 1993-1994 The Regents of the University of California.  
Copyright ' 1994-1996 Sun Microsystems, Inc.   
Copyright ' 1995-1997 Roger E. Critchlow Jr.
catch

NAME

catch – Evaluate script and trap exceptional returns

SYNOPSIS

catch script ?varName?

DESCRIPTION

The catch command may be used to prevent errors from aborting command interpretation. The catch command calls the Tcl interpreter recursively to execute script, and always returns without raising an error, regardless of any errors that might occur while executing script.

If script raises an error, catch will return a non-zero integer value corresponding to the exceptional return code returned by evaluation of script. Tcl defines the normal return code from script evaluation to be zero (0), or TCL_OK. Tcl also defines four exceptional return codes: 1 (TCL_ERROR), 2 (TCL_RETURN), 3 (TCL_BREAK), and 4 (TCL_CONTINUE). Errors during evaluation of a script are indicated by a return code of TCL_ERROR. The other exceptional return codes are returned by the return, break, and continue commands and in other special situations as documented. Tcl packages can define new commands that return other integer values as return codes as well, and scripts that make use of the return –code command can also have return codes other than the five defined by Tcl.

If the varName argument is given, then the variable it names is set to the result of the script evaluation. When the return code from the script is 1 (TCL_ERROR), the value stored in varName is an error message. When the return code from the script is 0 (TCL_OK), the value stored in resultVarName is the value returned from script.

If script does not raise an error, catch will return 0 (TCL_OK) and set the variable to the value returned from script.

Note that catch catches all exceptions, including those generated by break and continue as well as errors. The only errors that are not caught are syntax errors found when the script is compiled. This is because the catch command only catches errors during runtime. When the catch statement is compiled, the script is compiled as well and any syntax errors will generate a Tcl error.

EXAMPLES

The catch command may be used in an if to branch based on the success of a script.

```tcl
if { [catch {open $someFile w} fid] } {
    puts stderr "Could not open $someFile for writing\n$fid"
    exit 1
```
The `catch` command will not catch compiled syntax errors. The first time proc `foo` is called, the body will be compiled and a Tcl error will be generated.

```tcl
proc foo {} {
    catch {expr {1 +− }}
}
```

**SEE ALSO**

`break`, `continue`, `error`, `return`, `tclvars`

**KEYWORDS**

`catch`, `error`

---

*Copyright © 1993−1994 The Regents of the University of California.*  
*Copyright © 1994−1996 Sun Microsystems, Inc.*  
*Copyright © 1995−1997 Roger E. Critchlow Jr.*
cd

NAME

cd – Change working directory

SYNOPSIS

cd ?dirName?

DESCRIPTION

Change the current working directory to dirName, or to the home directory (as specified in the HOME environment variable) if dirName is not given. Returns an empty string. Note that the current working directory is a per-process resource; the cd command changes the working directory for all interpreters and (in a threaded environment) all threads.

EXAMPLES

Change to the home directory of the user fred:

    cd ~fred

Change to the directory lib that is a sibling directory of the current one:

    cd ../lib

SEE ALSO

filename, glob, pwd

KEYWORDS

working directory

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

clock – Obtain and manipulate time

SYNOPSIS

clock option ?arg arg ...?

description

clock clicks ?--milliseconds?
clock format clockValue ?--format string? ?--gmt boolean?

%% %a %A %b %B %c %C %d %D %e %g %G %h %H %I %j %k %l
%m %M %n %p %r %R %s %S %t %T %u %U %v %W %x
NAME

clock – Obtain and manipulate time

SYNOPSIS

clock option ?arg arg ...?

DESCRIPTION

This command performs one of several operations that may obtain or manipulate strings or values that represent some notion of time. The option argument determines what action is carried out by the command. The legal options (which may be abbreviated) are:

- **clock clicks ?-milliseconds?**
  Return a high-resolution time value as a system-dependent integer value. The unit of the value is system-dependent but should be the highest resolution clock available on the system such as a CPU cycle counter. If -milliseconds is specified, then the value is guaranteed to be of millisecond granularity. This value should only be used for the relative measurement of elapsed time.

- **clock format clockValue ?-format string? ?-gmt boolean?**
  Converts an integer time value, typically returned by clock seconds, clock scan, or the atime or mtime options of the file command, to human-readable form. If the -format argument is present the next argument is a string that describes how the date and time are to be formatted. Field descriptors consist of a % followed by a field descriptor character. All other characters are copied into the result. Valid field descriptors are:

  - %
  - Insert a %.

  - %a
  - Abbreviated weekday name (Mon, Tue, etc.).

  - %A
  - Full weekday name (Monday, Tuesday, etc.).
%b
Abbreviated month name (Jan, Feb, etc.).

%B
Full month name.

%c
Locale specific date and time. The format for date and time in the default "C" locale on Unix/Mac is "%a %b %d %H:%M:%S %Y". On Windows, this value is the locale specific long date and time, as specified in the Regional Options control panel settings.

%C
First two digits of the four−digit year (19 or 20).

%d
Day of month (01 − 31).

%D
Date as %m/%d/%y.

%e
Day of month (1 − 31), no leading zeros.

%g
The ISO8601 year number corresponding to the ISO8601 week (%V), expressed as a two−digit year−of−the−century, with leading zero if necessary.

%G
The ISO8601 year number corresponding to the ISO8601 week (%V), expressed as a four−digit number.

%h
Abbreviated month name.

%H
Hour in 24−hour format (00 − 23).

%I
Hour in 12−hour format (01 − 12).

%j
Day of year (001 − 366).

%k
Hour in 24−hour format, without leading zeros (0 − 23).

%l
Hour in 12−hour format, without leading zeros (1 − 12).

%m
Month number (01 − 12).

%M
Minute (00 − 59).

%n
Insert a newline.

%p
AM/PM indicator.

%r
Time in a locale−specific "meridian" format. The "meridian" format in the default "C" locale is "%I:%M:%S %p".

NAME
%R
Time as %H:%M.
%s
Count of seconds since the epoch, expressed as a decimal integer.
%S
Seconds (00 – 59).
%t
Insert a tab.
%T
Time as %H:%M:%S.
%u
Weekday number (Monday = 1, Sunday = 7).
%U
Week of year (00 – 52), Sunday is the first day of the week.
%V
Week of year according to ISO-8601 rules. Week 1 of a given year is the week containing 4 January.
%w
Weekday number (Sunday = 0, Saturday = 6).
%W
Week of year (00 – 52), Monday is the first day of the week.
%x
Locale specific date format. The format for a date in the default "C" locale for Unix/Mac is "%m/%d/%y". On Windows, this value is the locale specific short date format, as specified in the Regional Options control panel settings.
%X
Locale specific 24-hour time format. The format for a 24-hour time in the default "C" locale for Unix/Mac is "%H:%M:%S". On Windows, this value is the locale specific time format, as specified in the Regional Options control panel settings.
%y
Year without century (00 – 99).
%Y
Year with century (e.g. 1990)
%Z
Time zone name.

If the -format argument is not specified, the format string "%a %b %d %H:%M:%S %Z %Y" is used. If the -gmt argument is present the next argument must be a boolean which if true specifies that the time will be formatted as Greenwich Mean Time. If false then the local timezone will be used as defined by the operating environment.

clock scan dateString ?-base clockVal? ?-gmt boolean?
Convert dateString to an integer clock value (see clock seconds). This command can parse and convert virtually any standard date and/or time string, which can include standard time zone mnemonics. If only a time is specified, the current date is assumed. If the string does not contain a time zone mnemonic, the local time zone is assumed, unless the -gmt argument is true, in which case the clock value is calculated assuming that the specified time is relative to Greenwich Mean Time.
−gmt, if specified, affects only the computed time value; it does not impact the interpretation of −base.

If the −base flag is specified, the next argument should contain an integer clock value. Only the date in this value is used, not the time. This is useful for determining the time on a specific day or doing other date–relative conversions.

The dateString consists of zero or more specifications of the following form:

time
A time of day, which is of the form: hh:mm:ss?? ?? ?meridian? ?zone? or hhmm ?? ?? ?meridian? ?zone?. If no meridian is specified, hh is interpreted on a 24–hour clock.

date
A specific month and day with optional year. The acceptable formats are mm/dd/yy?, monthname dd ?, yy?, dd monthname yy?, day, dd monthname yy, ?CC?yyymmdd, ?CC?yy−mm−dd, dd–monthname–?CC?yy. The default year is the current year. If the year is less than 100, we treat the years 00–68 as 2000–2068 and the years 69–99 as 1969–1999. Not all platforms can represent the years 38–70, so an error may result if these years are used.

ISO 8601 point–in–time
An ISO 8601 point–in–time specification, such as CCyymmddThhmmss, where T is the literal T, CCyymmdd hhmmss, or CCyymmddThh:mm:ss.

relative time
A specification relative to the current time. The format is number unit acceptable units are year, fortnight, month, week, day, hour, minute (or min), and second (or sec). The unit can be specified as a singular or plural, as in 3 weeks. These modifiers may also be specified: tomorrow, yesterday, today, now, last, this, next, ago.

The actual date is calculated according to the following steps. First, any absolute date and/or time is processed and converted. Using that time as the base, day–of–week specifications are added. Next, relative specifications are used. If a date or day is specified, and no absolute or relative time is given, midnight is used. Finally, a correction is applied so that the correct hour of the day is produced after allowing for daylight savings time differences and the correct date is given when going from the end of a long month to a short month.

Daylight savings time correction is applied only when the relative time is specified in units of days or more, ie, days, weeks, fortnights, months or years. This means that when crossing the daylight savings time boundary, different results will be given for clock scan "1 day" and clock scan "24 hours":

% clock scan "1 day" −base [clock scan 1999−10−31] 941443200
% clock scan "24 hours" −base [clock scan 1999−10−31] 941439600

clock seconds
Return the current date and time as a system–dependent integer value. The unit of the value is seconds, allowing it to be used for relative time calculations. The value is usually defined as total
elapsed time from an `epoch`. You shouldn't assume the value of the epoch.

SEE ALSO

date, time

KEYWORDS

clock, date, time

Copyright ' 1992-1995 Karl Lehenbauer and Mark Diekhans.
Copyright ' 1995-1997 Sun Microsystems, Inc.
Copyright ' 1998-1999 Scriptics Corporation
Copyright ' 2002 ActiveState Corporation
Copyright ' 1995-1997 Roger E. Critchlow Jr.
close

NAME

close – Close an open channel.

SYNOPSIS

close channelId

DESCRIPTION

Closes the channel given by channelId.

ChannelId must be an identifier for an open channel such as a Tcl standard channel (stdin, stdout, or stderr), the return value from an invocation of open or socket, or the result of a channel creation command provided by a Tcl extension.

All buffered output is flushed to the channel's output device, any buffered input is discarded, the underlying file or device is closed, and channelId becomes unavailable for use.

If the channel is blocking, the command does not return until all output is flushed. If the channel is nonblocking and there is unflushed output, the channel remains open and the command returns immediately; output will be flushed in the background and the channel will be closed when all the flushing is complete.

If channelId is a blocking channel for a command pipeline then close waits for the child processes to complete.

If the channel is shared between interpreters, then close makes channelId unavailable in the invoking interpreter but has no other effect until all of the sharing interpreters have closed the channel. When the last interpreter in which the channel is registered invokes close, the cleanup actions described above occur. See the interp command for a description of channel sharing.

Channels are automatically closed when an interpreter is destroyed and when the process exits. Channels are switched to blocking mode, to ensure that all output is correctly flushed before the process exits.

The command returns an empty string, and may generate an error if an error occurs while flushing output. If a command in a command pipeline created with open returns an error, close generates an error (similar to the exec command.)

EXAMPLE

This illustrates how you can use Tcl to ensure that files get closed even when errors happen by combining catch, close and return:
proc withOpenFile {filename channelVar script} {
    upvar 1 $channelVar chan
    set chan [open $filename]
    catch {
        uplevel 1 $script
    } result options
    close $chan
    return -options $options $result
}

SEE ALSO

file, open, socket, eof, Tcl_StandardChannels

KEYWORDS

blocking, channel, close, nonblocking

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
concat

NAME

concat – Join lists together

SYNOPSIS

concat ?arg arg ...?

DESCRIPTION

This command joins each of its arguments together with spaces after trimming leading and trailing white-space from each of them. If all the arguments are lists, this has the same effect as concatenating them into a single list. It permits any number of arguments; if no args are supplied, the result is an empty string.

EXAMPLES

Although concat will concatenate lists (so the command:

concat a b {c d e} {f {g h}}

will return "a b c d e f {g h}" as its result), it will also concatenate things that are not lists, and hence the command:

concat "a b {c " d " e} f"

will return "a b {c d e} f" as its result.

Note that the concatenation does not remove spaces from the middle of its arguments, so the command:

concat "a b c { d e f }

will return "a b c d e f" (i.e. with three spaces between the a, the b and the c).

SEE ALSO

append, eval

KEYWORDS

concatenate, join, lists

Copyright © 1993 The Regents of the University of California.
concat
continue

NAME

continue – Skip to the next iteration of a loop

SYNOPSIS

continue

DESCRIPTION

This command is typically invoked inside the body of a looping command such as `for` or `foreach` or `while`. It returns a TCL_CONTINUE code, which causes a continue exception to occur. The exception causes the current script to be aborted out to the innermost containing loop command, which then continues with the next iteration of the loop. Catch exceptions are also handled in a few other situations, such as the `catch` command and the outermost scripts of procedure bodies.

EXAMPLE

Print a line for each of the integers from 0 to 10 except 5:

```tcl
for {set x 0} {$x<10} {incr x} {
    if {$x == 5} {
        continue
    } else {
        puts "x is $x"
    }
}
```

SEE ALSO

break, for, foreach, return, while

KEYWORDS

continue, iteration, loop
NAME

dde – Execute a Dynamic Data Exchange command

SYNOPSIS

package require dde 1.2

dde eval ?−async? service cmd ?arg ...?
dde execute ?−async? service topic data
dde poke service topic item data
dde request ?−binary? service topic data
dde servername ?topic?
dde services service topic

DESCRIPTION

DDE COMMANDS

dde servername ?topic?
dde execute ?−async? service topic data
dde poke service topic item data
dde request ?−binary? service topic item
dde services service topic
dde eval ?−async? topic cmd ?arg arg ...?

DDE AND TCL

EXAMPLE

SEE ALSO

KEYWORDS

NAME

dde – Execute a Dynamic Data Exchange command

SYNOPSIS

package require dde 1.2

dde eval ?−async? service cmd ?arg ...?
dde execute ?−async? service topic data
dde poke service topic item data
dde request ?−binary? service topic data
dde servername ?topic?
dde services service topic
DESCRIPTION

This command allows an application to send Dynamic Data Exchange (DDE) command when running under Microsoft Windows. Dynamic Data Exchange is a mechanism where applications can exchange raw data. Each DDE transaction needs a service name and a topic. Both the service name and topic are application defined; Tcl uses the service name TclEval, while the topic name is the name of the interpreter given by dde servername. Other applications have their own service names and topics. For instance, Microsoft Excel has the service name Excel.

The eval and execute commands accept the option –async:

DDE COMMANDS

The following commands are a subset of the full Dynamic Data Exchange set of commands.

dde servername ?topic?

dde servername registers the interpreter as a DDE server with the service name TclEval and the topic name specified by topic. If no topic is given, dde servername returns the name of the current topic or the empty string if it is not registered as a service.

dde execute ?–async? service topic data

dde execute takes the data and sends it to the server indicated by service with the topic indicated by topic. Typically, service is the name of an application, and topic is a file to work on. The data field is given to the remote application. Typically, the application treats the data field as a script, and the script is run in the application. The –async option requests asynchronous invocation. The command returns an error message if the script did not run, unless the –async flag was used, in which case the command returns immediately with no error.

dde poke service topic item data

dde poke passes the data to the server indicated by service using the topic and item specified. Typically, service is the name of an application. topic is application specific but can be a command to the server or the name of a file to work on. The item is also application specific and is often not used, but it must always be non–null. The data field is given to the remote application.

dde request ?–binary? service topic item

dde request is typically used to get the value of something; the value of a cell in Microsoft Excel or the text of a selection in Microsoft Word. service is typically the name of an application, topic is typically the name of the file, and item is application–specific. The command returns the value of item as defined in the application. Normally this is interpreted to be a string with terminating null. If –binary is specified, the result is returned as a byte array.

dde services service topic

dde services returns a list of service–topic pairs that currently exist on the machine. If service and topic are both null strings ({}), then all service–topic pairs currently available on the system are returned. If service is null and topic is not, then all services with the specified topic are returned. If service is not null and topic is, all topics for a given service are returned. If both are not null, if that service–topic pair currently exists, it is returned; otherwise, null is returned.

dde eval ?–async? topic cmd ?arg arg ...?
**dde eval** evaluates a command and its arguments using the interpreter specified by *topic*. The DDE service must be the **TclEval** service. The **–async** option requests asynchronous invocation. The command returns an error message if the script did not run, unless the **–async** flag was used, in which case the command returns immediately with no error. This command can be used to replace **send** on Windows.

**DDE AND TCL**

A Tcl interpreter always has a service name of **TclEval**. Each different interpreter of all running Tcl applications must be given a unique name specified by **dde servername**. Each interp is available as a DDE topic only if the **dde servername** command was used to set the name of the topic for each interp. So a **dde services TclEval {}** command will return a list of service−topic pairs, where each of the currently running interps will be a topic.

When Tcl processes a **dde execute** command, the data for the execute is run as a script in the interp named by the topic of the **dde execute** command.

When Tcl processes a **dde request** command, it returns the value of the variable given in the dde command in the context of the interp named by the dde topic. Tcl reserves the variable **$TCLEVAL$EXECUTE$RESULT** for internal use, and **dde request** commands for that variable will give unpredictable results.

An external application which wishes to run a script in Tcl should have that script store its result in a variable, run the **dde execute** command, and the run **dde request** to get the value of the variable.

When using DDE, be careful to ensure that the event queue is flushed using either **update** or **vwait**. This happens by default when using **wish** unless a blocking command is called (such as **exec** without adding the & to place the process in the background). If for any reason the event queue is not flushed, DDE commands may hang until the event queue is flushed. This can create a deadlock situation.

**EXAMPLE**

This asks Internet Explorer (which must already be running) to go to a particularly important website:

```tcl
package require dde
dde execute iexplore WWW_OpenURL http://www.tcl.tk/
```

**SEE ALSO**

**tk, wino, send**

**KEYWORDS**

application, dde, name, remote execution
encoding

NAME
encoding – Manipulate encodings

SYNOPSIS
encoding option ?arg arg ...?

DESCRIPTION
encoding convertfrom ?encoding? data
encoding convertto ?encoding? string
encoding names
encoding system ?encoding?

EXAMPLE

SEE ALSO

KEYWORDS
encoding is not specified, the current system encoding is used.

encoding names
Returns a list containing the names of all of the encodings that are currently available.

encoding system ?encoding?
Set the system encoding to encoding. If encoding is omitted then the command returns the current system encoding. The system encoding is used whenever Tcl passes strings to system calls.

EXAMPLE

It is common practice to write script files using a text editor that produces output in the euc−jp encoding, which represents the ASCII characters as single bytes and Japanese characters as two bytes. This makes it easy to embed literal strings that correspond to non−ASCII characters by simply typing the strings in place in the script. However, because the source command always reads files using the current system encoding, Tcl will only source such files correctly when the encoding used to write the file is the same. This tends not to be true in an internationalized setting. For example, if such a file was sourced in North America (where the ISO8859−1 is normally used), each byte in the file would be treated as a separate character that maps to the 00 page in Unicode. The resulting Tcl strings will not contain the expected Japanese characters. Instead, they will contain a sequence of Latin−1 characters that correspond to the bytes of the original string. The encoding command can be used to convert this string to the expected Japanese Unicode characters. For example,

```
set s [encoding convertfrom euc−jp "\xA4\xCF"]
```

would return the Unicode string "\u306F", which is the Hiragana letter HA.

SEE ALSO

Tcl_GetEncoding

KEYWORDS

encoding
eof

NAME

eof – Check for end of file condition on channel

SYNOPSIS

eof channelId

DESCRIPTION

Returns 1 if an end of file condition occurred during the most recent input operation on channelId (such as gets). 0 otherwise.

ChannelId must be an identifier for an open channel such as a Tcl standard channel (stdin, stdout, or stderr), the return value from an invocation of open or socket, or the result of a channel creation command provided by a Tcl extension.

EXAMPLES

Read and print out the contents of a file line–by–line:

```tcl
set f [open somefile.txt]
while {1} {
    set line [gets $f]
    if {[eof $f]} {
        close $f
        break
    }
    puts "Read line: $line"
}
```

Read and print out the contents of a file by fixed–size records:

```tcl
set f [open somefile.dat]
fconfigure $f -translation binary
set recordSize 40
while {1} {
    set record [read $f $recordSize]
    if {[eof $f]} {
        close $f
        break
    }
    puts "Read record: $record"
}
```
SEE ALSO

file, open, close, fblocked, Tcl StandardChannels

KEYWORDS

channel, end of file

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
error

NAME

error – Generate an error

SYNOPSIS

error message ?info? ?code?

DESCRIPTION

Returns a TCL_ERROR code, which causes command interpretation to be unwound. Message is a string that is returned to the application to indicate what went wrong.

If the info argument is provided and is non–empty, it is used to initialize the global variable errorInfo. errorInfo is used to accumulate a stack trace of what was in progress when an error occurred; as nested commands unwind, the Tcl interpreter adds information to errorInfo. If the info argument is present, it is used to initialize errorInfo and the first increment of unwind information will not be added by the Tcl interpreter. In other words, the command containing the error command will not appear in errorInfo; in its place will be info. This feature is most useful in conjunction with the catch command: if a caught error cannot be handled successfully, info can be used to return a stack trace reflecting the original point of occurrence of the error:

```tcl
catch {...} errMsg
set savedInfo $errorInfo
... 
error $errMsg $savedInfo
```

If the code argument is present, then its value is stored in the errorCode global variable. This variable is intended to hold a machine–readable description of the error in cases where such information is available; see the tclvars manual page for information on the proper format for the variable. If the code argument is not present, then errorCode is automatically reset to `"NONE" by the Tcl interpreter as part of processing the error generated by the command.

EXAMPLE

Generate an error if a basic mathematical operation fails:

```tcl
if {1+2 != 3} {
    error "something is very wrong with addition"
}
```
SEE ALSO

catch, return, tclvars

KEYWORDS

error, errorCode, errorInfo

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
eval

NAME
eval – Evaluate a Tcl script

SYNOPSIS
eval arg ?arg ...?

DESCRIPTION
Eval takes one or more arguments, which together comprise a Tcl script containing one or more commands. Eval concatenates all its arguments in the same fashion as the concat command, passes the concatenated string to the Tcl interpreter recursively, and returns the result of that evaluation (or any error generated by it). Note that the list command quotes sequences of words in such a way that they are not further expanded by the eval command.

EXAMPLE
This procedure acts in a way that is analogous to the lappend command, except it inserts the argument values at the start of the list in the variable:

```tcl
proc lprepend {varName args} {
    upvar 1 $varName var
    # Ensure that the variable exists and contains a list
    lappend var
    # Now we insert all the arguments in one go
    set var [eval [list linsert $var 0] $args]
}
```

KEYWORDS
concatenate, evaluate, script

SEE ALSO
catch, concat, error, list, subst, tclvars

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
exec

NAME

exec – Invoke subprocesses

SYNOPSIS

exec ?switches? arg ?arg ...?

DESCRIPTION

-keepnewline

==

|&

< fileName

<@ fileId

<< value

> fileName

2> fileName

>>& fileName

2>>& fileName

>>&& fileName

>@ fileId

2>@ fileId

>&@ fileId

PORTABILITY ISSUES

Windows (all versions)

Windows NT

Windows 9x

Macintosh

Unix

UNIX EXAMPLES

WINDOWS EXAMPLES

SEE ALSO

KEYWORDS

NAME

exec – Invoke subprocesses

SYNOPSIS

exec ?switches? arg ?arg ...?
DESCRIPTION

This command treats its arguments as the specification of one or more subprocesses to execute. The arguments take the form of a standard shell pipeline where each arg becomes one word of a command, and each distinct command becomes a subprocess.

If the initial arguments to exec start with -- then they are treated as command-line switches and are not part of the pipeline specification. The following switches are currently supported:

--keepnewline
  Retains a trailing newline in the pipeline's output. Normally a trailing newline will be deleted.

--
  Marks the end of switches. The argument following this one will be treated as the first arg even if it starts with a --.

If an arg (or pair of args) has one of the forms described below then it is used by exec to control the flow of input and output among the subprocess(es). Such arguments will not be passed to the subprocess(es). In forms such as ``< fileName" fileName may either be in a separate argument from "<" or in the same argument with no intervening space (i.e. "<fileName").

|  Separates distinct commands in the pipeline. The standard output of the preceding command will be piped into the standard input of the next command.

|&  Separates distinct commands in the pipeline. Both standard output and standard error of the preceding command will be piped into the standard input of the next command. This form of redirection overrides forms such as 2> and >&.

< fileName
  The file named by fileName is opened and used as the standard input for the first command in the pipeline.

<@ fileId
  FileId must be the identifier for an open file, such as the return value from a previous call to open. It is used as the standard input for the first command in the pipeline. FileId must have been opened for reading.

<< value
  Value is passed to the first command as its standard input.

> fileName
  Standard output from the last command is redirected to the file named fileName, overwriting its previous contents.

2> fileName
  Standard error from all commands in the pipeline is redirected to the file named fileName, overwriting its previous contents.

>& fileName
  Both standard output from the last command and standard error from all commands are redirected to the file named fileName, overwriting its previous contents.
>> fileName
  Standard output from the last command is redirected to the file named fileName, appending to it rather than overwriting it.

2>> fileName
  Standard error from all commands in the pipeline is redirected to the file named fileName, appending to it rather than overwriting it.

>>>& fileName
  Both standard output from the last command and standard error from all commands are redirected to the file named fileName, appending to it rather than overwriting it.

>@ fileId
  FileId must be the identifier for an open file, such as the return value from a previous call to open.
  Standard output from the last command is redirected to fileId's file, which must have been opened for writing.

2>@ fileId
  FileId must be the identifier for an open file, such as the return value from a previous call to open.
  Standard error from all commands in the pipeline is redirected to fileId's file. The file must have been opened for writing.

>&@ fileId
  FileId must be the identifier for an open file, such as the return value from a previous call to open.
  Both standard output from the last command and standard error from all commands are redirected to fileId's file. The file must have been opened for writing.

If standard output has not been redirected then the exec command returns the standard output from the last command in the pipeline. If any of the commands in the pipeline exit abnormally or are killed or suspended, then exec will return an error and the error message will include the pipeline's output followed by error messages describing the abnormal terminations; the errorCode variable will contain additional information about the last abnormal termination encountered. If any of the commands writes to its standard error file and that standard error isn't redirected, then exec will return an error; the error message will include the pipeline's standard output, followed by messages about abnormal terminations (if any), followed by the standard error output.

If the last character of the result or error message is a newline then that character is normally deleted from the result or error message. This is consistent with other Tcl return values, which don't normally end with newlines. However, if --keepnewline is specified then the trailing newline is retained.

If standard input isn't redirected with ``<" or ``<<" or ``<@" then the standard input for the first command in the pipeline is taken from the application's current standard input.

If the last arg is ``&" then the pipeline will be executed in background. In this case the exec command will return a list whose elements are the process identifiers for all of the subprocesses in the pipeline. The standard output from the last command in the pipeline will go to the application's standard output if it hasn't been redirected, and error output from all of the commands in the pipeline will go to the application's standard error file unless redirected.

The first word in each command is taken as the command name; tilde-substitution is performed on it, and if
the result contains no slashes then the directories in the PATH environment variable are searched for an executable by the given name. If the name contains a slash then it must refer to an executable reachable from the current directory. No "glob" expansion or other shell–like substitutions are performed on the arguments to commands.

**PORTABILITY ISSUES**

**Windows (all versions)**

Reading from or writing to a socket, using the "`@ fileId`" notation, does not work. When reading from a socket, a 16–bit DOS application will hang and a 32–bit application will return immediately with end–of–file. When either type of application writes to a socket, the information is instead sent to the console, if one is present, or is discarded.

The Tk console text widget does not provide real standard IO capabilities. Under Tk, when redirecting from standard input, all applications will see an immediate end–of–file; information redirected to standard output or standard error will be discarded.

Either forward or backward slashes are accepted as path separators for arguments to Tcl commands. When executing an application, the path name specified for the application may also contain forward or backward slashes as path separators. Bear in mind, however, that most Windows applications accept arguments with forward slashes only as option delimiters and backslashes only in paths. Any arguments to an application that specify a path name with forward slashes will not automatically be converted to use the backslash character. If an argument contains forward slashes as the path separator, it may or may not be recognized as a path name, depending on the program.

Additionally, when calling a 16–bit DOS or Windows 3.X application, all path names must use the short, cryptic, path format (e.g., using `applba~1.def` instead of `applbakery.default`), which can be obtained with the `file attributes $fileName −shortname` command.

Two or more forward or backward slashes in a row in a path refer to a network path. For example, a simple concatenation of the root directory `c:/` with a subdirectory `/windows/system` will yield `c://windows/system` (two slashes together), which refers to the mount point called system on the machine called windows (and the `c:/` is ignored), and is not equivalent to `c:/windows/system`, which describes a directory on the current computer. The `file join` command should be used to concatenate path components.

Note that there are two general types of Win32 console applications:

1) CLI — CommandLine Interface, simple stdio exchange. `netstat.exe` for example.
2) TUI — Textmode User Interface, any application that accesses the console API for doing such things as cursor movement, setting text color, detecting key presses and mouse movement, etc. An example would be `telnet.exe` from Windows 2000. These types of applications are not common in a windows environment, but do exist.

`exec` will not work well with TUI applications when a console is not present, as is done when launching applications under wish. It is desirable to have console applications hidden and
detached. This is a designed-in limitation as exec wants to communicate over pipes. The Expect extension addresses this issue when communicating with a TUI application.

**Windows NT**

When attempting to execute an application, exec first searches for the name as it was specified. Then, in order, .com, .exe, and .bat are appended to the end of the specified name and it searches for the longer name. If a directory name was not specified as part of the application name, the following directories are automatically searched in order when attempting to locate the application:

- The directory from which the Tcl executable was loaded.
- The current directory.
- The Windows NT 32-bit system directory.
- The Windows NT 16-bit system directory.
- The Windows NT home directory.
- The directories listed in the path.

In order to execute shell built-in commands like dir and copy, the caller must prepend the desired command with `"cmd.exe /c "` because built-in commands are not implemented using executables.

**Windows 9x**

When attempting to execute an application, exec first searches for the name as it was specified. Then, in order, .com, .exe, and .bat are appended to the end of the specified name and it searches for the longer name. If a directory name was not specified as part of the application name, the following directories are automatically searched in order when attempting to locate the application:

- The directory from which the Tcl executable was loaded.
- The current directory.
- The Windows 9x system directory.
- The Windows 9x home directory.
- The directories listed in the path.

In order to execute shell built-in commands like dir and copy, the caller must prepend the desired command with `"command.com /c "` because built-in commands are not implemented using executables.

Once a 16-bit DOS application has read standard input from a console and then quit, all subsequently run 16-bit DOS applications will see the standard input as already closed. 32-bit applications do not have this problem and will run correctly, even after a 16-bit DOS application thinks that standard input is closed. There is no known workaround for this bug at this time.

Redirection between the NUL: device and a 16-bit application does not always work. When redirecting from NUL:, some applications may hang, others will get an infinite stream of "0x01" bytes, and some will actually correctly get an immediate end-of-file; the behavior seems to depend upon something compiled into the application itself. When redirecting greater than 4K or so to NUL:, some applications will hang. The above problems do not happen with 32-bit applications.

All DOS 16-bit applications are run synchronously. All standard input from a pipe to a 16-bit DOS application is collected into a temporary file; the other end of the pipe must be closed before the 16-bit DOS application begins executing. All standard output or error from a 16-bit DOS application...
to a pipe is collected into temporary files; the application must terminate before the temporary files are redirected to the next stage of the pipeline. This is due to a workaround for a Windows 95 bug in the implementation of pipes, and is how the standard Windows 95 DOS shell handles pipes itself.

Certain applications, such as `command.com`, should not be executed interactively. Applications which directly access the console window, rather than reading from their standard input and writing to their standard output may fail, hang Tcl, or even hang the system if their own private console window is not available to them.

**Macintosh**

The `exec` command is not implemented and does not exist under Macintosh.

**Unix**

The `exec` command is fully functional and works as described.

**UNIX EXAMPLES**

Here are some examples of the use of the `exec` command on Unix.

To execute a simple program and get its result:

```
exec uname -a
```

To execute a program that can return a non–zero result, you should wrap the call to `exec` in `catch` and check what the contents of the global `errorCode` variable is if you have an error:

```
set status 0
if {[catch {exec grep foo bar.txt} results]} {
    if {[lindex ::errorCode 0] eq "CHILDSTATUS"} {
        set status [lindex ::errorCode 2]
    } else {
        # Some kind of unexpected failure
    }
}
```

When translating a command from a Unix shell invocation, care should be taken over the fact that single quote characters have no special significance to Tcl. Thus:

```
awk '{sum += $1} END {print sum}' numbers.list
```

would be translated into something like:

```
exec awk {{sum += $1} END {print sum}} numbers.list
```

If you are converting invocations involving shell globbing, you should remember that Tcl does not handle globbing or expand things into multiple arguments by default. Instead you should write things like this:

```
eval [list exec ls -l] [glob *.tcl]
```
WINDOWS EXAMPLES

Here are some examples of the use of the exec command on Windows.

To start an instance of notepad editing a file without waiting for the user to finish editing the file:

```
exec notepad myfile.txt &
```

To print a text file using notepad:

```
exec notepad /p myfile.txt
```

If a program calls other programs, such as is common with compilers, then you may need to resort to batch files to hide the console windows that sometimes pop up:

```
exec cmp.bat somefile.c -o somefile
```

With the file cmp.bat looking something like:

```
@gcc %1 %2 %3 %4 %5 %6 %7 %8 %9
```

Sometimes you need to be careful, as different programs may have the same name and be in the path. It can then happen that typing a command at the DOS prompt finds a different program than the same command run via exec. This is because of the (documented) differences in behaviour between exec and DOS batch files.

When in doubt, use the command auto_execok: it will return the complete path to the program as seen by the exec command. This applies especially when you want to run "internal" commands like dir from a Tcl script (if you just want to list filenames, use the glob command.) To do that, use this:

```
eval [list exec] [auto_execok dir] [list *.tcl]
```

SEE ALSO

error, open

KEYWORDS

execute, pipeline, redirection, subprocess
exit

NAME

exit – End the application

SYNOPSIS

exit ?returnCode?

DESCRIPTION

Terminate the process, returning returnCode to the system as the exit status. If returnCode isn't specified then it defaults to 0.

EXAMPLE

Since non–zero exit codes are usually interpreted as error cases by the calling process, the exit command is an important part of signalling that something fatal has gone wrong. This code fragment is useful in scripts to act as a general problem trap:

```tcl
proc main {} {
    # ... put the real main code in here ...
}
if {[catch {main} msg]} {
    puts stderr "unexpected script error: $msg"
    if {[info exist env(DEBUG)]} {
        puts stderr "----- BEGIN TRACE ----"
        puts stderr $errorInfo
        puts stderr "----- END TRACE ----"
    }  
    # Reserve code 1 for "expected" error exits...
    exit 2
}
```

SEE ALSO

exec, tclvars

KEYWORDS

exit, process

Copyright © 1993 The Regents of the University of California.
NAME
expr – Evaluate an expression

SYNOPSIS
expr arg ?arg arg ...?

DESCRIPTION
OPERANDS
OPERATORS
= + ~ !
* / %
+ -
<< >>
< > <= >=
== !=
eq ne
&
^ |
&& \\
?:
MATH FUNCTIONS
abs(arg)
acos(arg)
asin(arg)
atan(arg)
atan2(y, x)
ceil(arg)
cos(arg)
cosh(arg)
double(arg)
ext(arg)
floor(arg)
fmod(x, y)
hypot(x, y)
int(x, y)
log(arg)
log10(arg)
pow(x, y)
rand()
round(arg)
sin(arg)
```plaintext
sinh(arg)
sqrt(arg)
srand(arg)
tan(arg)
tanh(arg)
wide(arg)
```

TYPES, OVERFLOW, AND PRECISION
STRING OPERATIONS
PERFORMANCE CONSIDERATIONS
EXAMPLES
SEE ALSO
KEYWORDS

NAME

expr – Evaluate an expression

SYNOPSIS

```plaintext
expr arg ?arg arg ...?
```

DESCRIPTION

Concatenates `args` (adding separator spaces between them), evaluates the result as a Tcl expression, and returns the value. The operators permitted in Tcl expressions are a subset of the operators permitted in C expressions, and they have the same meaning and precedence as the corresponding C operators. Expressions almost always yield numeric results (integer or floating-point values). For example, the expression

```plaintext
expr 8.2 + 6
```

evaluates to 14.2. Tcl expressions differ from C expressions in the way that operands are specified. Also, Tcl expressions support non–numeric operands and string comparisons.

OPERANDS

A Tcl expression consists of a combination of operands, operators, and parentheses. White space may be used between the operands and operators and parentheses; it is ignored by the expression's instructions. Where possible, operands are interpreted as integer values. Integer values may be specified in decimal (the normal case), in octal (if the first character of the operand is 0), or in hexadecimal (if the first two characters of the operand are 0x). If an operand does not have one of the integer formats given above, then it is treated as a floating–point number if that is possible. Floating–point numbers may be specified in any of the ways accepted by an ANSI–compliant C compiler (except that the f, F, l, and L suffixes will not be permitted in most installations). For example, all of the following are valid floating–point numbers: 2.1, 3., 6e4, 7.91e+16. If no numeric interpretation is possible (note that all literal operands that are not numeric or boolean must be quoted with either braces or with double quotes), then an operand is left as a string (and only a limited set of
```
operators may be applied to it).

On 32-bit systems, integer values MAX_INT (0x7FFFFFFF) and MIN_INT (−0x80000000) will be represented as 32-bit values, and integer values outside that range will be represented as 64-bit values (if that is possible at all.)

Operands may be specified in any of the following ways:

1. As a numeric value, either integer or floating-point.
2. As a boolean value, using any form understood by `string is boolean`.
3. As a Tcl variable, using standard `$` notation. The variable's value will be used as the operand.
4. As a string enclosed in double-quotes. The expression parser will perform backslash, variable, and command substitutions on the information between the quotes, and use the resulting value as the operand.
5. As a string enclosed in braces. The characters between the open brace and matching close brace will be used as the operand without any substitutions.
6. As a Tcl command enclosed in brackets. The command will be executed and its result will be used as the operand.
7. As a mathematical function whose arguments have any of the above forms for operands, such as `sin($x)`. See below for a list of defined functions.

Where the above substitutions occur (e.g. inside quoted strings), they are performed by the expression's instructions. However, the command parser may already have performed one round of substitution before the expression processor was called. As discussed below, it is usually best to enclose expressions in braces to prevent the command parser from performing substitutions on the contents.

For some examples of simple expressions, suppose the variable `a` has the value 3 and the variable `b` has the value 6. Then the command on the left side of each of the lines below will produce the value on the right side of the line:

```
expr 3.1 + $a   6.1
expr 2 + "$a.$b"  5.6
expr 4*[llength "6 2"]  8
expr {word one} < "word $a"  0
```

**OPERATORS**

The valid operators are listed below, grouped in decreasing order of precedence:
Unary minus, unary plus, bit-wise NOT, logical NOT. None of these operators may be applied to string operands, and bit-wise NOT may be applied only to integers.

Multiply, divide, remainder. None of these operators may be applied to string operands, and remainder may be applied only to integers. The remainder will always have the same sign as the divisor and an absolute value smaller than the divisor.

Add and subtract. Valid for any numeric operands.

Left and right shift. Valid for integer operands only. A right shift always propagates the sign bit.

Boolean less, greater, less than or equal, and greater than or equal. Each operator produces 1 if the condition is true, 0 otherwise. These operators may be applied to strings as well as numeric operands, in which case string comparison is used.

Boolean equal and not equal. Each operator produces a zero/one result. Valid for all operand types.

Boolean string equal and string not equal. Each operator produces a zero/one result. The operand types are interpreted only as strings.

Bit-wise AND. Valid for integer operands only.

Bit-wise exclusive OR. Valid for integer operands only.

Bit-wise OR. Valid for integer operands only.

Logical AND. Produces a 1 result if both operands are non-zero, 0 otherwise. Valid for boolean and numeric (integers or floating-point) operands only.

Logical OR. Produces a 0 result if both operands are zero, 1 otherwise. Valid for boolean and numeric (integers or floating-point) operands only.

If–then–else, as in C. If \( x \) evaluates to non-zero, then the result is the value of \( y \). Otherwise the result is the value of \( z \). The \( x \) operand must have a boolean or numeric value.

See the C manual for more details on the results produced by each operator. All of the binary operators group left–to–right within the same precedence level. For example, the command

\[
\text{expr } 4*2 < 7
\]

returns 0.

The `&&`, `||`, and `?:` operators have ``lazy evaluation``, just as in C, which means that operands are not evaluated if they are not needed to determine the outcome. For example, in the command
expr {$v ? [a] : [b]}

only one of [a] or [b] will actually be evaluated, depending on the value of $v. Note, however, that this is only true if the entire expression is enclosed in braces; otherwise the Tcl parser will evaluate both [a] and [b] before invoking the expr command.

MATH FUNCTIONS

Tcl supports the following mathematical functions in expressions, all of which work solely with floating-point numbers unless otherwise noted:

```
abs(arg)
acos(arg)
asin(arg)
atan(arg)
atan2(y, x)
ceil(arg)
cos(arg)
cosh(arg)
double(arg)
exp(arg)
floor(arg)
fmod(x, y)
```

- **abs**(arg)
  Returns the absolute value of arg. Arg may be either integer or floating-point, and the result is returned in the same form.

- **acos**(arg)
  Returns the arc cosine of arg, in the range [0,pi] radians. Arg should be in the range [-1,1].

- **asin**(arg)
  Returns the arc sine of arg, in the range [-pi/2,pi/2] radians. Arg should be in the range [-1,1].

- **atan**(arg)
  Returns the arc tangent of arg, in the range [-pi/2,pi/2] radians.

- **atan2**(y, x)
  Returns the arc tangent of y/x, in the range [-pi,pi] radians. x and y cannot both be 0. If x is greater than 0, this is equivalent to **atan**(y/x).

- **ceil**(arg)
  Returns the smallest integral floating-point value (i.e. with a zero fractional part) not less than arg.

- **cos**(arg)
  Returns the cosine of arg, measured in radians.

- **cosh**(arg)
  Returns the hyperbolic cosine of arg. If the result would cause an overflow, an error is returned.

- **double**(arg)
  If arg is a floating-point value, returns arg, otherwise converts arg to floating-point and returns the converted value.

- **exp**(arg)
  Returns the exponential of arg, defined as e**arg. If the result would cause an overflow, an error is returned.

- **floor**(arg)
  Returns the largest integral floating-point value (i.e. with a zero fractional part) not greater than arg.

- **fmod**(x, y)
  Returns the floating-point remainder of the division of x by y. If y is 0, an error is returned.
**hypot***(x, y)**
Computes the length of the hypotenuse of a right–angled triangle \( \sqrt{x^2 + y^2} \).

**int**(arg)
If arg is an integer value of the same width as the machine word, returns arg, otherwise converts arg to an integer (of the same size as a machine word, i.e. 32–bits on 32–bit systems, and 64–bits on 64–bit systems) by truncation and returns the converted value.

**log**(arg)
Returns the natural logarithm of arg. Arg must be a positive value.

**log10**(arg)
Returns the base 10 logarithm of arg. Arg must be a positive value.

**pow**(x, y)
Computes the value of \( x \) raised to the power \( y \). If \( x \) is negative, \( y \) must be an integer value.

**rand()**
Returns a pseudo–random floating–point value in the range \((0,1)\). The generator algorithm is a simple linear congruential generator that is not cryptographically secure. Each result from rand completely determines all future results from subsequent calls to rand, so rand should not be used to generate a sequence of secrets, such as one–time passwords. The seed of the generator is initialized from the internal clock of the machine or may be set with the srand function.

**round**(arg)
If arg is an integer value, returns arg, otherwise converts arg to integer by rounding and returns the converted value.

**sin**(arg)
Returns the sine of arg, measured in radians.

**sinh**(arg)
Returns the hyperbolic sine of arg. If the result would cause an overflow, an error is returned.

**sqrt**(arg)
Returns the square root of arg. Arg must be non–negative.

**srand**(arg)
The arg, which must be an integer, is used to reset the seed for the random number generator of rand. Returns the first random number (see rand()) from that seed. Each interpreter has its own seed.

**tan**(arg)
Returns the tangent of arg, measured in radians.

**tanh**(arg)
Returns the hyperbolic tangent of arg.

**wide**(arg)
Converts arg to an integer value at least 64–bits wide (by sign–extension if arg is a 32–bit number) if it is not one already.

In addition to these predefined functions, applications may define additional functions using 

**Tcl_CreateMathFunc**().

**TYPES, OVERFLOW, AND PRECISION**

All internal computations involving integers are done with the C type long, and all internal computations involving floating–point are done with the C type double. When converting a string to floating–point,
exponent overflow is detected and results in a Tcl error. For conversion to integer from string, detection of overflow depends on the behavior of some routines in the local C library, so it should be regarded as unreliable. In any case, integer overflow and underflow are generally not detected reliably for intermediate results. Floating–point overflow and underflow are detected to the degree supported by the hardware, which is generally pretty reliable.

Conversion among internal representations for integer, floating–point, and string operands is done automatically as needed. For arithmetic computations, integers are used until some floating–point number is introduced, after which floating–point is used. For example,

```
expr 5 / 4
```

returns 1, while

```
expr 5 / 4.0
expr 5 / ( [string length "abcd"] + 0.0 )
```

both return 1.25. Floating–point values are always returned with a `.`` or an `e` so that they will not look like integer values. For example,

```
expr 20.0/5.0
```

returns 4.0, not 4.

**STRING OPERATIONS**

String values may be used as operands of the comparison operators, although the expression evaluator tries to do comparisons as integer or floating–point when it can, except in the case of the `eq` and `ne` operators. If one of the operands of a comparison is a string and the other has a numeric value, the numeric operand is converted back to a string using the C `sprintf` format specifier `%d` for integers and `%g` for floating–point values. For example, the commands

```
expr {"0x03" > "2"}
expr {"0y" < "0x12"}
```

both return 1. The first comparison is done using integer comparison, and the second is done using string comparison after the second operand is converted to the string 18. Because of Tcl's tendency to treat values as numbers whenever possible, it isn't generally a good idea to use operators like `==` when you really want string comparison and the values of the operands could be arbitrary; it's better in these cases to use the `eq` or `ne` operators, or the `string` command instead.

**PERFORMANCE CONSIDERATIONS**

Enclose expressions in braces for the best speed and the smallest storage requirements. This allows the Tcl bytecode compiler to generate the best code.
As mentioned above, expressions are substituted twice: once by the Tcl parser and once by the `expr` command. For example, the commands

```tcl
set a 3
set b {\$a + 2}
expr \$b*4
```

return 11, not a multiple of 4. This is because the Tcl parser will first substitute \$a + 2 for the variable \$b, then the `expr` command will evaluate the expression \$a + 2*4.

Most expressions do not require a second round of substitutions. Either they are enclosed in braces or, if not, their variable and command substitutions yield numbers or strings that don't themselves require substitutions. However, because a few unbraced expressions need two rounds of substitutions, the bytecode compiler must emit additional instructions to handle this situation. The most expensive code is required for unbraced expressions that contain command substitutions. These expressions must be implemented by generating new code each time the expression is executed.

**EXAMPLES**

Define a procedure that computes an "interesting" mathematical function:

```tcl
proc calc {x y} {
    expr { ($x*$x − $y*$y) / exp($x*$x + $y*$y) }
}
```

Convert polar coordinates into cartesian coordinates:

```tcl
# convert from ($radius,$angle)
set x [expr { $radius * cos($angle) }]
set y [expr { $radius * sin($angle) }]
```

Convert cartesian coordinates into polar coordinates:

```tcl
# convert from ($x,$y)
set radius [expr { hypot($y, $x) }]
set angle  [expr { atan2($y, $x) }]
```

Print a message describing the relationship of two string values to each other:

```tcl
puts "a and b are [expr \$a eq \$b ? {equal} : {different}]"
```

Set a variable to whether an environment variable is both defined at all and also set to a true boolean value:

```tcl
set isTrue [expr {
    [info exists ::env(SOME_ENV_VAR)] &&
    [string is true −strict ::env(SOME_ENV_VAR)]
}]
```
Generate a random integer in the range 0..99 inclusive:

```tcl
set randNum [expr { int(100 * rand()) }]
```

**SEE ALSO**

*array, for, if, string, Tcl, while*

**KEYWORDS**

*arithmetic, boolean, compare, expression, fuzzy comparison*
NAME

fblocked – Test whether the last input operation exhausted all available input

SYNOPSIS

fblocked channelId

DESCRIPTION

The fblocked command returns 1 if the most recent input operation on channelId returned less information than requested because all available input was exhausted. For example, if gets is invoked when there are only three characters available for input and no end-of-line sequence, gets returns an empty string and a subsequent call to fblocked will return 1.

ChannelId must be an identifier for an open channel such as a Tcl standard channel (stdin, stdout, or stderr), the return value from an invocation of open or socket, or the result of a channel creation command provided by a Tcl extension.

EXAMPLE

The fblocked command is particularly useful when writing network servers, as it allows you to write your code in a line-by-line style without preventing the servicing of other connections. This can be seen in this simple echo-service:

```tcl
# This is called whenever a new client connects to the server
proc connect {chan host port} {
    set clientName [format <%s:%d> $host $port]
    puts "connection from $clientName"
    fconfigure $chan −blocking 0 −buffering line
    fileevent $chan readable [list echoLine $chan $clientName]
}

# This is called whenever either at least one byte of input
# data is available, or the channel was closed by the client.
proc echoLine {chan clientName} {
    gets $chan line
    if {[eof $chan]} {
        puts "finishing connection from $clientName"
        close $chan
    } elseif {![fblocked $chan]} {
        # Didn't block waiting for end-of-line
        puts "$clientName - $line"
        puts $chan $line
    }
}
```

fblocked
# Create the server socket and enter the event-loop to wait
# for incoming connections...
socket -server connect 12345
vwait forever

SEE ALSO

gets, open, read, socket, Tcl_StandardChannels

KEYWORDS

blocking, nonblocking

Copyright ' 1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

fconfigure – Set and get options on a channel

SYNOPSIS

fconfigure channelId
fconfigure channelId name
fconfigure channelId name value ?name value ...?

DESCRIPTION

−blocking boolean
−buffering newValue
−buffersize newSize
−encoding name
−eofchar char
−eofchar {inChar outChar}
−translation mode
−translation {inMode outMode}
  auto
  binary
  cr
  crlf
  lf

STANDARD CHANNELS

EXAMPLES

SEE ALSO

KEYWORDS

NAME

fconfigure – Set and get options on a channel

SYNOPSIS

fconfigure channelId
fconfigure channelId name
fconfigure channelId name value ?name value ...?

DESCRIPTION

The fconfigure command sets and retrieves options for channels.
ChannelId identifies the channel for which to set or query an option and must refer to an open channel such as a Tcl standard channel (stdin, stdout, or stderr), the return value from an invocation of open or socket, or the result of a channel creation command provided by a Tcl extension.

If no name or value arguments are supplied, the command returns a list containing alternating option names and values for the channel. If name is supplied but no value then the command returns the current value of the given option. If one or more pairs of name and value are supplied, the command sets each of the named options to the corresponding value; in this case the return value is an empty string.

The options described below are supported for all channels. In addition, each channel type may add options that only it supports. See the manual entry for the command that creates each type of channels for the options that that specific type of channel supports. For example, see the manual entry for the socket command for its additional options.

−blocking boolean
The −blocking option determines whether I/O operations on the channel can cause the process to block indefinitely. The value of the option must be a proper boolean value. Channels are normally in blocking mode; if a channel is placed into nonblocking mode it will affect the operation of the gets, read, puts, flush, and close commands; see the documentation for those commands for details. For nonblocking mode to work correctly, the application must be using the Tcl event loop (e.g. by calling Tcl_DoOneEvent or invoking the vwait command).

−buffering newValue
If newValue is full then the I/O system will buffer output until its internal buffer is full or until the flush command is invoked. If newValue is line, then the I/O system will automatically flush output for the channel whenever a newline character is output. If newValue is none, the I/O system will flush automatically after every output operation. The default is for −buffering to be set to full except for channels that connect to terminal–like devices; for these channels the initial setting is line. Additionally, stdin and stdout are initially set to line, and stderr is set to none.

−buffersize newSize
Newsize must be an integer; its value is used to set the size of buffers, in bytes, subsequently allocated for this channel to store input or output. Newsize must be between ten and one million, allowing buffers of ten to one million bytes in size.

−encoding name
This option is used to specify the encoding of the channel, so that the data can be converted to and from Unicode for use in Tcl. For instance, in order for Tcl to read characters from a Japanese file in shiftjis and properly process and display the contents, the encoding would be set to shiftjis. Thereafter, when reading from the channel, the bytes in the Japanese file would be converted to Unicode as they are read. Writing is also supported – as Tcl strings are written to the channel they will automatically be converted to the specified encoding on output.

If a file contains pure binary data (for instance, a JPEG image), the encoding for the channel should be configured to be binary. Tcl will then assign no interpretation to the data in the file and simply read or write raw bytes. The Tcl binary command can be used to manipulate this byte–oriented data.

The default encoding for newly opened channels is the same platform– and locale–dependent system encoding as for the current console window.
encoding used for interfacing with the operating system.

\texttt{−eofchar char}
\texttt{−eofchar \{inChar outChar\}}

This option supports DOS file systems that use Control−z (\texttt{\x1a}) as an end of file marker. If \texttt{char} is not an empty string, then this character signals end−of−file when it is encountered during input. For output, the end−of−file character is output when the channel is closed. If \texttt{char} is the empty string, then there is no special end of file character marker. For read−write channels, a two−element list specifies the end of file marker for input and output, respectively. As a convenience, when setting the end−of−file character for a read−write channel you can specify a single value that will apply to both reading and writing. When querying the end−of−file character of a read−write channel, a two−element list will always be returned. The default value for \texttt{−eofchar} is the empty string in all cases except for files under Windows. In that case the \texttt{−eofchar} is Control−z (\texttt{\x1a}) for reading and the empty string for writing.

\texttt{−translation mode}
\texttt{−translation \{inMode outMode\}}

In Tcl scripts the end of a line is always represented using a single newline character (\texttt{\n}). However, in actual files and devices the end of a line may be represented differently on different platforms, or even for different devices on the same platform. For example, under UNIX newlines are used in files, whereas carriage−return−linefeed sequences are normally used in network connections. On input (i.e., with \texttt{gets} and \texttt{read}) the Tcl I/O system automatically translates the external end−of−line representation into newline characters. Upon output (i.e., with \texttt{puts}), the I/O system translates newlines to the external end−of−line representation. The default translation mode, \texttt{auto}, handles all the common cases automatically, but the \texttt{−translation} option provides explicit control over the end of line translations.

The value associated with \texttt{−translation} is a single item for read−only and write−only channels. The value is a two−element list for read−write channels; the read translation mode is the first element of the list, and the write translation mode is the second element. As a convenience, when setting the translation mode for a read−write channel you can specify a single value that will apply to both reading and writing. When querying the translation mode of a read−write channel, a two−element list will always be returned. The following values are currently supported:

\texttt{auto}

As the input translation mode, \texttt{auto} treats any of newline (\texttt{lf}), carriage return (\texttt{cr}), or carriage return followed by a newline (\texttt{crlf}) as the end of line representation. The end of line representation can even change from line−to−line, and all cases are translated to a newline. As the output translation mode, \texttt{auto} chooses a platform specific representation; for sockets on all platforms Tcl chooses \texttt{crlf}, for all Unix flavors, it chooses \texttt{lf}, for the Macintosh platform it chooses \texttt{cr} and for the various flavors of Windows it chooses \texttt{crlf}. The default setting for \texttt{−translation} is \texttt{auto} for both input and output.

\texttt{binary}

No end−of−line translations are performed. This is nearly identical to \texttt{lf} mode, except that in addition \texttt{binary} mode also sets the end−of−file character to the empty string (which disables it) and sets the encoding to \texttt{binary} (which disables encoding filtering). See the description of \texttt{−eofchar} and \texttt{−encoding} for more information.
The end of a line in the underlying file or device is represented by a single carriage return character. As the input translation mode, **cr** mode converts carriage returns to newline characters. As the output translation mode, **cr** mode translates newline characters to carriage returns. This mode is typically used on Macintosh platforms.

**crlf**

The end of a line in the underlying file or device is represented by a carriage return character followed by a linefeed character. As the input translation mode, **crlf** mode converts carriage-return-linefeed sequences to newline characters. As the output translation mode, **crlf** mode translates newline characters to carriage-return-linefeed sequences. This mode is typically used on Windows platforms and for network connections.

**lf**

The end of a line in the underlying file or device is represented by a single newline (linefeed) character. In this mode no translations occur during either input or output. This mode is typically used on UNIX platforms.

### STANDARD CHANNELS

The Tcl standard channels (**stdin**, **stdout**, and **stderr**) can be configured through this command like every other channel opened by the Tcl library. Beyond the standard options described above they will also support any special option according to their current type. If, for example, a Tcl application is started by the **inet** super-server common on Unix system its Tcl standard channels will be sockets and thus support the socket options.

### EXAMPLES

Instruct Tcl to always send output to **stdout** immediately, whether or not it is to a terminal:

```
fconfigure stdout -buffering none
```

Open a socket and read lines from it without ever blocking the processing of other events:

```
set s [socket some.where.com 12345]
fconfigure $s -blocking 0
fileevent $s readable "readMe $s"
proc readMe chan {
  if {[gets $chan line] < 0} {
    if {[eof $chan]} {
      close $chan
      return
    } # Could not read a complete line this time; Tcl's
    # internal buffering will hold the partial line for us
    # until some more data is available over the socket.
  } else {
    puts stdout $line
  }
}
```
Read a PPM-format image from a file:

```tcl
# Open the file and put it into Unix ASCII mode
set f [open teapot.ppm]
  
  # Get the header
  if {[gets $f] ne "P6"} {
    error "not a raw-bits PPM"
  }

  # Read lines until we have got non-comment lines
  # that supply us with three decimal values.
  set words {}
  while {[llength $words] < 3} {
    gets $f line
    if {[string match "#*" $line]} continue
    lappend words [eval concat [scan $line %d%d%d]]
  }

  # Those words supply the size of the image and its
  # overall depth per channel. Assign to variables.
  foreach {xSize ySize depth} $words {break}

  # Now switch to binary mode to pull in the data,
  # one byte per channel (red,green,blue) per pixel.
  fconfigure $f -translation binary
  set numDataBytes [expr {3 * $xSize * $ySize}]
  set data [read $f $numDataBytes]

  close $f
```

SEE ALSO

`close`, `flush`, `gets`, `open`, `puts`, `read`, `socket`, `Tcl_StandardChannels`

KEYWORDS

`blocking`, `buffering`, `carriage return`, `end of line`, `flushing`, `linemode`, `newline`, `nonblocking`, `platform`, `translation`, `encoding`, `filter`, `byte array`, `binary`

---

Copyright ' 1995-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
fcopy

NAME

fcopy – Copy data from one channel to another.

SYNOPSIS

fcopy inchan outchan ?–size size? ?–command callback?

DESCRIPTION

The fcopy command copies data from one I/O channel, inchan to another I/O channel, outchan. The fcopy command leverages the buffering in the Tcl I/O system to avoid extra copies and to avoid buffering too much data in main memory when copying large files to slow destinations like network sockets.

The fcopy command transfers data from inchan until end of file or size bytes have been transferred. If no –size argument is given, then the copy goes until end of file. All the data read from inchan is copied to outchan. Without the –command option, fcopy blocks until the copy is complete and returns the number of bytes written to outchan.

The –command argument makes fcopy work in the background. In this case it returns immediately and the callback is invoked later when the copy completes. The callback is called with one or two additional arguments that indicates how many bytes were written to outchan. If an error occurred during the background copy, the second argument is the error string associated with the error. With a background copy, it is not necessary to put inchan or outchan into non-blocking mode; the fcopy command takes care of that automatically. However, it is necessary to enter the event loop by using the vwait command or by using Tk.

You are not allowed to do other I/O operations with inchan or outchan during a background fcopy. If either inchan or outchan get closed while the copy is in progress, the current copy is stopped and the command callback is not made. If inchan is closed, then all data already queued for outchan is written out.

Note that inchan can become readable during a background copy. You should turn off any fileevent handlers during a background copy so those handlers do not interfere with the copy. Any I/O attempted by a fileevent handler will get a "channel busy" error.

Fcopy translates end–of–line sequences in inchan and outchan according to the –translation option for these channels. See the manual entry for configure for details on the –translation option. The translations mean that the number of bytes read from inchan can be different than the number of bytes written to outchan. Only the number of bytes written to outchan is reported, either as the return value of a synchronous fcopy or as the argument to the callback for an asynchronous fcopy.

Fcopy obeys the encodings configured for the channels. This means that the incoming characters are converted internally first UTF–8 and then into the encoding of the channel fcopy writes to. See the manual
entry for `fconfigure` for details on the `−encoding` option. No conversion is done if both channels are set to encoding "binary". If only the output channel is set to encoding "binary" the system will write the internal UTF-8 representation of the incoming characters. If only the input channel is set to encoding "binary" the system will assume that the incoming bytes are valid UTF-8 characters and convert them according to the output encoding. The behaviour of the system for bytes which are not valid UTF-8 characters is undefined in this case.

EXAMPLE

This first example shows how the callback gets passed the number of bytes transferred. It also uses `vwait` to put the application into the event loop. Of course, this simplified example could be done without the command callback.

```tcl
proc Cleanup {in out bytes {error {}}} {
    global total
    set total $bytes
    close $in
    close $out
    if {([string length $error] != 0) { # error occurred during the copy}
    }
}
set in [open $file1]
set out [socket $server $port]
fcopy $in $out −command [list Cleanup $in $out]
vwait total
```

The second example copies in chunks and tests for end of file in the command callback

```tcl
proc CopyMore {in out chunk bytes {error {}}} {
    global total done
    incr total $bytes
    if {([string length $error] != 0) || [eof $in] { 
        set done $total
        close $in
        close $out
    } else {
        fcopy $in $out −command [list CopyMore $in $out $chunk] \
        −size $chunk
    }
}
set in [open $file1]
set out [socket $server $port]
set chunk 1024
set total 0
fcopy $in $out −command [list CopyMore $in $out $chunk] −size $chunk
vwait done
```
SEE ALSO

cnf, fblocked, fconfigure

KEYWORDS

blocking, channel, end of line, end of file, nonblocking, read, translation

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1997 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

file – Manipulate file names and attributes

SYNOPSIS

file option name ?arg arg ...?

DESCRIPTION

file atime name ?time?
file attributes name
file attributes name ?option?
file attributes name ?option value option value...?
file channels ?pattern?
file copy ?force? ?–−? source target
file dirname name
file executable name
file exists name
file extension name
file isdirectory name
file isfile name
file join name ?name ...?
file link ?−linktype? linkName ?target?
file lstat name varName
file mkdir dir ?dir ...?
file mtime name ?time?
file nativename name
file normalize name
file owned name
file pathtype name
file readable name
file readlink name
file rename ?force? ?–−? source target
file rootname name
file separator ?name?
file size name
file split name
file stat name varName
file system name
file tail name
file type name
file volumes
**NAME**

file – Manipulate file names and attributes

**SYNOPSIS**

file option name ?arg arg ...?

**DESCRIPTION**

This command provides several operations on a file's name or attributes. *Name* is the name of a file; if it starts with a tilde, then tilde substitution is done before executing the command (see the manual entry for [filename](#) for details). *Option* indicates what to do with the file name. Any unique abbreviation for *option* is acceptable. The valid options are:

*file atime name ?time?*

Returns a decimal string giving the time at which file *name* was last accessed. If *time* is specified, it is an access time to set for the file. The time is measured in the standard POSIX fashion as seconds from a fixed starting time (often January 1, 1970). If the file doesn't exist or its access time cannot be queried or set then an error is generated. On Windows, FAT file systems do not support access time.

*file attributes name*

*file attributes name ?option?*

*file attributes name ?option value option value...?*

This subcommand returns or sets platform specific values associated with a file. The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:

On Unix, *−group* gets or sets the group name for the file. A group id can be given to the command, but it returns a group name. *−owner* gets or sets the user name of the owner of the file. The command returns the owner name, but the numerical id can be passed when setting the owner. *−permissions* sets or retrieves the octal code that chmod(1) uses. This command does also have limited support for setting using the symbolic attributes for chmod(1), of the form [ugo]?[+−=][rwxst], where multiple symbolic attributes can be separated by commas (example: *u+s,go−rw* add sticky bit for user, remove read and write permissions for group and other). A simplified *ls* style string, of the form rwxrwxrwx (must be 9 characters), is also supported (example: *rwxr−xr−t* is equivalent to 01755).
On Windows, --archive gives the value or sets or clears the archive attribute of the file. --hidden gives the value or sets or clears the hidden attribute of the file. --longname will expand each path element to its long version. This attribute cannot be set. --readonly gives the value or sets or clears the readonly attribute of the file. --shortname gives a string where every path element is replaced with its short (8.3) version of the name. This attribute cannot be set. --system gives or sets or clears the value of the system attribute of the file.

On Macintosh, --creator gives or sets the Finder creator type of the file. --hidden gives or sets or clears the hidden attribute of the file. --readonly gives or sets or clears the readonly attribute of the file. Note that directories can only be locked if File Sharing is turned on. --type gives or sets the Finder file type for the file.

file channels ?pattern?
If pattern isn't specified, returns a list of names of all registered open channels in this interpreter. If pattern is specified, only those names matching pattern are returned. Matching is determined using the same rules as for string match.

file copy ?--force ? ?-- ? source target
file copy ?--force ? ?-- ? source ?source ... ? targetDir
The first form makes a copy of the file or directory source under the pathname target. If target is an existing directory, then the second form is used. The second form makes a copy inside targetDir of each source file listed. If a directory is specified as a source, then the contents of the directory will be recursively copied into targetDir. Existing files will not be overwritten unless the --force option is specified. When copying within a single filesystem, file copy will copy soft links (i.e. the links themselves are copied, not the things they point to). Trying to overwrite a non-empty directory, overwrite a directory with a file, or overwrite a file with a directory will all result in errors even if --force was specified. Arguments are processed in the order specified, halting at the first error, if any. A -- marks the end of switches; the argument following the -- will be treated as a source even if it starts with a --.

file delete ?--force ? ?-- ? pathname ?pathname ... ?
Removes the file or directory specified by each pathname argument. Non-empty directories will be removed only if the --force option is specified. When operating on symbolic links, the links themselves will be deleted, not the objects they point to. Trying to delete a non-existent file is not considered an error. Trying to delete a read-only file will cause the file to be deleted, even if the --force flags is not specified. If the --force option is specified on a directory, Tcl will attempt both to change permissions and move the current directory 'pwd' out of the given path if that is necessary to allow the deletion to proceed. Arguments are processed in the order specified, halting at the first error, if any. A -- marks the end of switches; the argument following the -- will be treated as a pathname even if it starts with a --.

file dirname name
Returns a name comprised of all of the path components in name excluding the last element. If name is a relative file name and only contains one path element, then returns `"" (or ":" on the Macintosh). If name refers to a root directory, then the root directory is returned. For example,

file dirname c:/
returns c:/.
Note that tilde substitution will only be performed if it is necessary to complete the command. For example,

\texttt{file dirname ~/src/foo.c}

returns \texttt{/src}, whereas

\texttt{file dirname ~}

returns \texttt{/home} (or something similar).

\textbf{file executable name}

Returns 1 if file \texttt{name} is executable by the current user, 0 otherwise.

\textbf{file exists name}

Returns 1 if file \texttt{name} exists and the current user has search privileges for the directories leading to it, 0 otherwise.

\textbf{file extension name}

Returns all of the characters in \texttt{name} after and including the last dot in the last element of \texttt{name}. If there is no dot in the last element of \texttt{name} then returns the empty string.

\textbf{file isdirectory name}

Returns 1 if file \texttt{name} is a directory, 0 otherwise.

\textbf{file isfile name}

Returns 1 if file \texttt{name} is a regular file, 0 otherwise.

\textbf{file join name ?name ...?}

Takes one or more file names and combines them, using the correct path separator for the current platform. If a particular \texttt{name} is relative, then it will be joined to the previous file name argument. Otherwise, any earlier arguments will be discarded, and joining will proceed from the current argument. For example,

\texttt{file join a b /foo bar}

returns \texttt{/foo/bar}.

Note that any of the names can contain separators, and that the result is always canonical for the current platform: / for Unix and Windows, and : for Macintosh.

\textbf{file link ?−linktype? linkName ?target?}

If only one argument is given, that argument is assumed to be \texttt{linkName}, and this command returns the value of the link given by \texttt{linkName} (i.e. the name of the file it points to). If \texttt{linkName} isn't a link or its value cannot be read (as, for example, seems to be the case with hard links, which look just like ordinary files), then an error is returned. If 2 arguments are given, then these are assumed to be \texttt{linkName} and \texttt{target}. If \texttt{linkName} already exists, or if \texttt{target} doesn't exist, an error will be returned. Otherwise, Tcl creates a new link called \texttt{linkName} which points to the existing filesystem object at \texttt{target}, where the type of the link is platform-specific (on Unix a symbolic link will be the default).

This is useful for the case where the user wishes to create a link in a cross-platform way, and doesn't care what type of link is created. If the user wishes to make a link of a specific type only, (and signal an error if for some reason that is not possible), then the optional \texttt{−linktype} argument should be given. Accepted values for \texttt{−linktype} are "−symbolic" and "−hard". When creating links on filesystems that
either do not support any links, or do not support the specific type requested, an error message will be returned. In particular Windows 95, 98 and ME do not support any links at present, but most Unix platforms support both symbolic and hard links (the latter for files only), MacOS supports symbolic links and Windows NT/2000/XP (on NTFS drives) support symbolic directory links and hard file links.

```tcl
file lstat name varName
```

Same as `stat` option (see below) except uses the `lstat` kernel call instead of `stat`. This means that if `name` refers to a symbolic link the information returned in `varName` is for the link rather than the file it refers to. On systems that don't support symbolic links this option behaves exactly the same as the `stat` option.

```tcl
file mkdir dir ?dir ...?
```

Creates each directory specified. For each pathname `dir` specified, this command will create all non-existing parent directories as well as `dir` itself. If an existing directory is specified, then no action is taken and no error is returned. Trying to overwrite an existing file with a directory will result in an error. Arguments are processed in the order specified, halting at the first error, if any.

```tcl
file mtime name ?time?
```

Returns a decimal string giving the time at which file `name` was last modified. If `time` is specified, it is a modification time to set for the file (equivalent to Unix `touch`). The time is measured in the standard POSIX fashion as seconds from a fixed starting time (often January 1, 1970). If the file doesn't exist or its modified time cannot be queried or set then an error is generated.

```tcl
file nativename name
```

Returns the platform−specific name of the file. This is useful if the filename is needed to pass to a platform−specific call, such as exec under Windows or AppleScript on the Macintosh.

```tcl
file normalize name
```

Returns a unique normalized path representation for the file−system object (file, directory, link, etc), whose string value can be used as a unique identifier for it. A normalized path is an absolute path which has all '../', './' removed. Also it is one which is in the `standard` format for the native platform. On MacOS, Unix, this means the segments leading up to the path must be free of symbolic links/aliases (but the very last path component may be a symbolic link), and on Windows it also means we want the long form with that form's case−dependence (which gives us a unique, case−dependent path). The one exception concerning the last link in the path is necessary, because Tcl or the user may wish to operate on the actual symbolic link itself (for example 'file delete', 'file rename', 'file copy' are defined to operate on symbolic links, not on the things that they point to).

```tcl
file owned name
```

Returns 1 if file `name` is owned by the current user, 0 otherwise.

```tcl
file pathype name
```

Returns one of `absolute`, `relative`, `volumerelative`. If `name` refers to a specific file on a specific volume, the path type will be `absolute`. If `name` refers to a file relative to the current working directory, then the path type will be `relative`. If `name` refers to a file relative to the current working directory on a specified volume, or to a specific file on the current working volume, then the file type is `volumerelative`.

```tcl
file readable name
```

Returns 1 if file `name` is readable by the current user, 0 otherwise.

```tcl
file readlink name
```
Returns the value of the symbolic link given by name (i.e. the name of the file it points to). If name isn't a symbolic link or its value cannot be read, then an error is returned. On systems that don't support symbolic links this option is undefined.


The first form takes the file or directory specified by pathname source and renames it to target, moving the file if the pathname target specifies a name in a different directory. If target is an existing directory, then the second form is used. The second form moves each source file or directory into the directory targetDir. Existing files will not be overwritten unless the −force option is specified. When operating inside a single filesystem, Tcl will rename symbolic links rather than the things that they point to. Trying to overwrite a non-empty directory, overwrite a directory with a file, or a file with a directory will all result in errors. Arguments are processed in the order specified, halting at the first error, if any. A −− marks the end of switches; the argument following the −− will be treated as a source even if it starts with a −.

**file rootname** name

Returns all of the characters in name up to but not including the last `.` character in the last component of name. If the last component of name doesn't contain a dot, then returns name.

**file separator** ?name?

If no argument is given, returns the character which is used to separate path segments for native files on this platform. If a path is given, the filesystem responsible for that path is asked to return its separator character. If no file system accepts name, an error is generated.

**file size** name

Returns a decimal string giving the size of file name in bytes. If the file doesn't exist or its size cannot be queried then an error is generated.

**file split** name

Returns a list whose elements are the path components in name. The first element of the list will have the same path type as name. All other elements will be relative. Path separators will be discarded unless they are needed ensure that an element is unambiguously relative. For example, under Unix

```
file split /foo/~bar/baz
```

returns / foo ~/bar baz to ensure that later commands that use the third component do not attempt to perform tilde substitution.

**file stat** name varName

Invokes the stat kernel call on name, and uses the variable given by varName to hold information returned from the kernel call. VarName is treated as an array variable, and the following elements of that variable are set: atime, etime, dev, gid, ino, mode, nlink, size, type, uid. Each element except type is a decimal string with the value of the corresponding field from the stat return structure; see the manual entry for stat for details on the meanings of the values. The type element gives the type of the file in the same form returned by the command file type. This command returns an empty string.

**file system** name

Returns a list of two elements, the first of which is the name of the filesystem to use for the file, and the second an arbitrary string representing the filesystem–specific nature or type of the location within that filesystem. If a filesystem only supports one type of file, the second element may be null.
For example the native files have a first element 'native', and a second element which is a platform–specific type name for the file's system (e.g. 'NTFS', 'FAT', etc), or possibly the empty string if no further information is available or if this is not implemented. A generic virtual file system might return the list 'vfs ftp' to represent a file on a remote ftp site mounted as a virtual filesystem through an extension called 'vfs'. If the file does not belong to any filesystem, an error is generated.

**file tail** name

Returns all of the characters in name after the last directory separator. If name contains no separators then returns name.

**file type** name

Returns a string giving the type of file name, which will be one of file, directory, characterSpecial, blockSpecial, fifo, link, or socket.

**file volumes**

Returns the absolute paths to the volumes mounted on the system, as a proper Tcl list. On the Macintosh, this will be a list of the mounted drives, both local and network. N.B. if two drives have the same name, they will both appear on the volume list, but there is currently no way, from Tcl, to access any but the first of these drives. On UNIX, the command will always return "/", since all filesystems are locally mounted. On Windows, it will return a list of the available local drives (e.g. {a:/ c:/}).

**file writable** name

Returns 1 if file name is writable by the current user, 0 otherwise.

**PORTABILITY ISSUES**

Unix

These commands always operate using the real user and group identifiers, not the effective ones.

**EXAMPLES**

This procedure shows how to search for C files in a given directory that have a correspondingly–named object file in the current directory:

```tcl
proc findMatchingCFiles {dir} {
    set files {}
    switch $::tcl_platform(platform) {
        windows {
            set ext .obj
        }
        unix {
            set ext .o
        }
    }
    foreach file [glob −nocomplain −directory $dir *.c] {
        set objectFile [file tail [file rootname $file]]$ext
        if {[file exists $objectFile]} {
            lappend files $file
        }
    }
    return $files
}
```
Rename a file and leave a symbolic link pointing from the old location to the new place:

```tcl
set oldName foobar.txt
set newName foo/bar.txt
# Make sure that where we're going to move to exists...
if {![file isdirectory [file dirname $newName]]} {
    file mkdir [file dirname $newName]
}
file rename $oldName $newName
file link -symbolic $oldName $newName
```

**SEE ALSO**

filename, open, close, eof, gets, tell, seek, fblocked, flush

**KEYWORDS**

attributes, copy files, delete files, directory, file, move files, name, rename files, stat

---

*Copyright ' 1993 The Regents of the University of California.*

*Copyright ' 1994-1996 Sun Microsystems, Inc.*

*Copyright ' 1995-1997 Roger E. Critchlow Jr.*
fileevent

NAME

fileevent – Execute a script when a channel becomes readable or writable

SYNOPSIS

fileevent channelId readable ?script?
fileevent channelId writable ?script?

DESCRIPTION

This command is used to create file event handlers. A file event handler is a binding between a channel and a script, such that the script is evaluated whenever the channel becomes readable or writable. File event handlers are most commonly used to allow data to be received from another process on an event−driven basis, so that the receiver can continue to interact with the user while waiting for the data to arrive. If an application invokes gets or read on a blocking channel when there is no input data available, the process will block; until the input data arrives, it will not be able to service other events, so it will appear to the user to ```freeze up```. With fileevent, the process can tell when data is present and only invoke gets or read when they won't block.

The channelId argument to fileevent refers to an open channel such as a Tcl standard channel (stdin, stdout, or stderr), the return value from an invocation of open or socket, or the result of a channel creation command provided by a Tcl extension.

If the script argument is specified, then fileevent creates a new event handler: script will be evaluated whenever the channel becomes readable or writable (depending on the second argument to fileevent). In this case fileevent returns an empty string. The readable and writable event handlers for a file are independent, and may be created and deleted separately. However, there may be at most one readable and one writable handler for a file at a given time in a given interpreter. If fileevent is called when the specified handler already exists in the invoking interpreter, the new script replaces the old one.

If the script argument is not specified, fileevent returns the current script for channelId, or an empty string if there is none. If the script argument is specified as an empty string then the event handler is deleted, so that no script will be invoked. A file event handler is also deleted automatically whenever its channel is closed or its interpreter is deleted.

A channel is considered to be readable if there is unread data available on the underlying device. A channel is also considered to be readable if there is unread data in an input buffer, except in the special case where the most recent attempt to read from the channel was a gets call that could not find a complete line in the input buffer. This feature allows a file to be read a line at a time in nonblocking mode using events. A channel is also considered to be readable if an end of file or error condition is present on the underlying file or device. It is important for script to check for these conditions and handle them appropriately; for example, if there is no special check for end of file, an infinite loop may occur where script reads no data, returns, and is
immediately invoked again.

A channel is considered to be writable if at least one byte of data can be written to the underlying file or device without blocking, or if an error condition is present on the underlying file or device.

Event-driven I/O works best for channels that have been placed into nonblocking mode with the \texttt{fconfigure} command. In blocking mode, a \texttt{puts} command may block if you give it more data than the underlying file or device can accept, and a \texttt{gets} or \texttt{read} command will block if you attempt to read more data than is ready; no events will be processed while the commands block. In nonblocking mode \texttt{puts}, \texttt{read}, and \texttt{gets} never block. See the documentation for the individual commands for information on how they handle blocking and nonblocking channels.

The script for a file event is executed at global level (outside the context of any Tcl procedure) in the interpreter in which the \texttt{fileevent} command was invoked. If an error occurs while executing the script then the \texttt{bgerror} mechanism is used to report the error. In addition, the file event handler is deleted if it ever returns an error; this is done in order to prevent infinite loops due to buggy handlers.

**EXAMPLE**

In this setup \texttt{GetData} will be called with the channel as an argument whenever \$chan becomes readable.

```tcl
proc GetData {chan} {
    if {![eof $chan]} {
        puts [gets $chan]
    }
}

fileevent $chan readable [list GetData $chan]
```

**CREDITS**

\texttt{fileevent} is based on the \texttt{addinput} command created by Mark Diekhans.

**SEE ALSO**

\texttt{bgerror, fconfigure, gets, puts, read, Tcl_StandardChannels}

**KEYWORDS**

asynchronous I/O, blocking, channel, event handler, nonblocking, readable, script, writable.
filename

NAME

filename – File name conventions supported by Tcl commands

INTRODUCTION
PATH TYPES
PATH SYNTAX

mac

:\ MyFile
:MyDisk:MyFile
:MyDir:MyFile
::MyFile
:::MyFile
/MyDisk/MyFile
../MyFile

unix

/ /etc/passwd
: foo
:foo/bar
:.foo

windows

\Host\share/file
c:foo
c:/foo
foo\bar
\foo
\foo

TILDE SUBSTITUTION
PORTABILITY ISSUES
KEYWORDS
SEE ALSO

NAME

filename – File name conventions supported by Tcl commands
INTRODUCTION

All Tcl commands and C procedures that take file names as arguments expect the file names to be in one of three forms, depending on the current platform. On each platform, Tcl supports file names in the standard forms(s) for that platform. In addition, on all platforms, Tcl supports a Unix−like syntax intended to provide a convenient way of constructing simple file names. However, scripts that are intended to be portable should not assume a particular form for file names. Instead, portable scripts must use the file split and file join commands to manipulate file names (see the file manual entry for more details).

PATH TYPES

File names are grouped into three general types based on the starting point for the path used to specify the file: absolute, relative, and volume−relative. Absolute names are completely qualified, giving a path to the file relative to a particular volume and the root directory on that volume. Relative names are unqualified, giving a path to the file relative to the current working directory. Volume−relative names are partially qualified, either giving the path relative to the root directory on the current volume, or relative to the current directory of the specified volume. The file pathtype command can be used to determine the type of a given path.

PATH SYNTAX

The rules for native names depend on the value reported in the Tcl array element tcl_platform(platform):

mac

On Apple Macintosh systems, Tcl supports two forms of path names. The normal Mac style names use colons as path separators. Paths may be relative or absolute, and file names may contain any character other than colon. A leading colon causes the rest of the path to be interpreted relative to the current directory. If a path contains a colon that is not at the beginning, then the path is interpreted as an absolute path. Sequences of two or more colons anywhere in the path are used to construct relative paths where :: refers to the parent of the current directory, :::: refers to the parent of the parent, and so forth.

In addition to Macintosh style names, Tcl also supports a subset of Unix−like names. If a path contains no colons, then it is interpreted like a Unix path. Slash is used as the path separator. The file name . refers to the current directory, and .. refers to the parent of the current directory. However, some names like / or /.. have no mapping, and are interpreted as Macintosh names. In general, commands that generate file names will return Macintosh style names, but commands that accept file names will take both Macintosh and Unix−style names.

The following examples illustrate various forms of path names:

:\nRelative path to the current folder.
MyFile
Relative path to a file named MyFile in the current folder.
MyDisk:MyFile
Absolute path to a file named MyFile on the device named MyDisk.

:MyDir:MyFile
Relative path to a file name MyFile in a folder named MyDir in the current folder.

::MyFile
Relative path to a file named MyFile in the folder above the current folder.

:::MyFile
Relative path to a file named MyFile in the folder two levels above the current folder.

/MyDisk/MyFile
Absolute path to a file named MyFile on the device named MyDisk.

../MyFile
Relative path to a file named MyFile in the folder above the current folder.

unix
On Unix platforms, Tcl uses path names where the components are separated by slashes. Path names may be relative or absolute, and file names may contain any character other than slash. The file names . and .. are special and refer to the current directory and the parent of the current directory respectively. Multiple adjacent slash characters are interpreted as a single separator. The following examples illustrate various forms of path names:

/ Absolute path to the root directory.

/etc/passwd
Absolute path to the file named passwd in the directory etc in the root directory.

.
Relative path to the current directory.

foo
Relative path to the file foo in the current directory.

foo/bar
Relative path to the file bar in the directory foo in the current directory.

../foo
Relative path to the file foo in the folder above the current directory.

windows
On Microsoft Windows platforms, Tcl supports both drive-relative and UNC style names. Both / and \ may be used as directory separators in either type of name. Drive-relative names consist of an optional drive specifier followed by an absolute or relative path. UNC paths follow the general form \servername\sharename\path\file, but must at the very least contain the server and share components, i.e. \servername\sharename. In both forms, the file names . and .. are special and refer to the current directory and the parent of the current directory respectively. The following examples illustrate various forms of path names:

\Host\share\file
Absolute UNC path to a file called file in the root directory of the export point share on the host Host. Note that repeated use of file dirname on this path will give //Host/share, and will never give just /fB//Host/fR.

c:foo
Volume-relative path to a file foo in the current directory on drive c.
c:/foo

Absolute path to a file foo in the root directory of drive c.

foo/bar

Relative path to a file bar in the foo directory in the current directory on the current volume.

\foo

Volume-relative path to a file foo in the root directory of the current volume.

\foo

Volume-relative path to a file foo in the root directory of the current volume. This is not a valid UNC path, so the assumption is that the extra backslashes are superfluous.

TILDE SUBSTITUTION

In addition to the file name rules described above, Tcl also supports csh-style tilde substitution. If a file name starts with a tilde, then the file name will be interpreted as if the first element is replaced with the location of the home directory for the given user. If the tilde is followed immediately by a separator, then the $HOME environment variable is substituted. Otherwise the characters between the tilde and the next separator are taken as a user name, which is used to retrieve the user's home directory for substitution.

The Macintosh and Windows platforms do not support tilde substitution when a user name follows the tilde. On these platforms, attempts to use a tilde followed by a user name will generate an error that the user does not exist when Tcl attempts to interpret that part of the path or otherwise access the file. The behaviour of these paths when not trying to interpret them is the same as on Unix. File names that have a tilde without a user name will be correctly substituted using the $HOME environment variable, just like for Unix.

PORTABILITY ISSUES

Not all file systems are case sensitive, so scripts should avoid code that depends on the case of characters in a file name. In addition, the character sets allowed on different devices may differ, so scripts should choose file names that do not contain special characters like: <>:"/\|. The safest approach is to use names consisting of alphanumeric characters only. Also Windows 3.1 only supports file names with a root of no more than 8 characters and an extension of no more than 3 characters.

On Windows platforms there are file and path length restrictions. Complete paths or filenames longer than about 260 characters will lead to errors in most file operations.

Another Windows peculiarity is that any number of trailing dots '.' in filenames are totally ignored, so, for example, attempts to create a file or directory with a name "foo." will result in the creation of a file/directory with name "foo". This fact is reflected in the results of 'file normalize'. Furthermore, a file name consisting only of dots '.........' or dots with trailing characters '.....abc' is illegal.

KEYWORDS

current directory, absolute file name, relative file name, volume-relative file name, portability
SEE ALSO

`file`, `glob`
flush

NAME

flush – Flush buffered output for a channel

SYNOPSIS

flush channelId

DESCRIPTION

Flushes any output that has been buffered for channelId.

ChannelId must be an identifier for an open channel such as a Tcl standard channel (stdout or stderr), the return value from an invocation of open or socket, or the result of a channel creation command provided by a Tcl extension. The channel must have been opened for writing.

If the channel is in blocking mode the command does not return until all the buffered output has been flushed to the channel. If the channel is in nonblocking mode, the command may return before all buffered output has been flushed; the remainder will be flushed in the background as fast as the underlying file or device is able to absorb it.

EXAMPLE

Prompt for the user to type some information in on the console:

puts -nonewline "Please type your name: "
flush stdout
gets stdin name
puts "Hello there, $name!"

SEE ALSO

file, open, socket, Tcl_StandardChannels

KEYWORDS

blocking, buffer, channel, flush, nonblocking, output

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
for

NAME

for – ``For'' loop

SYNOPSIS

for start test next body

DESCRIPTION

For is a looping command, similar in structure to the C for statement. The start, next, and body arguments must be Tcl command strings, and test is an expression string. The for command first invokes the Tcl interpreter to execute start. Then it repeatedly evaluates test as an expression; if the result is non-zero it invokes the Tcl interpreter on body, then invokes the Tcl interpreter on next, then repeats the loop. The command terminates when test evaluates to 0. If a continue command is invoked within body then any remaining commands in the current execution of body are skipped; processing continues by invoking the Tcl interpreter on next, then evaluating test, and so on. If a break command is invoked within body or next, then the for command will return immediately. The operation of break and continue are similar to the corresponding statements in C. For returns an empty string.

Note: test should almost always be enclosed in braces. If not, variable substitutions will be made before the for command starts executing, which means that variable changes made by the loop body will not be considered in the expression. This is likely to result in an infinite loop. If test is enclosed in braces, variable substitutions are delayed until the expression is evaluated (before each loop iteration), so changes in the variables will be visible. See below for an example:

EXAMPLES

Print a line for each of the integers from 0 to 10:

```tcl
for {set x 0} {$x<10} {incr x} {
    puts "x is $x"
}
```

Either loop infinitely or not at all because the expression being evaluated is actually the constant, or even generate an error! The actual behaviour will depend on whether the variable x exists before the for command is run and whether its value is a value that is less than or greater than/equal to ten, and this is because the expression will be substituted before the for command is executed.

```tcl
for {set x 0} {$x<10} {incr x} {
    puts "x is $x"
}
```
Print out the powers of two from 1 to 1024:

```tcl
for {set x 1} {$x<=1024} {set x [expr {$x * 2}]} {
    puts "x is $x"
}
```

SEE ALSO

`break`, `continue`, `foreach`, `while`

KEYWORDS

`for`, `iteration`, `looping`

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1997 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
foreach

NAME

foreach – Iterate over all elements in one or more lists

SYNOPSIS

foreach varname list body
foreach varlist1 list1 ?varlist2 list2 ...? body

DESCRIPTION

The `foreach` command implements a loop where the loop variable(s) take on values from one or more lists. In the simplest case there is one loop variable, `varname`, and one list, `list`, that is a list of values to assign to `varname`. The `body` argument is a Tcl script. For each element of `list` (in order from first to last), `foreach` assigns the contents of the element to `varname` as if the `lindex` command had been used to extract the element, then calls the Tcl interpreter to execute `body`.

In the general case there can be more than one value list (e.g., `list1` and `list2`), and each value list can be associated with a list of loop variables (e.g., `varlist1` and `varlist2`). During each iteration of the loop the variables of each `varlist` are assigned consecutive values from the corresponding `list`. Values in each `list` are used in order from first to last, and each value is used exactly once. The total number of loop iterations is large enough to use up all the values from all the value lists. If a value list does not contain enough elements for each of its loop variables in each iteration, empty values are used for the missing elements.

The `break` and `continue` statements may be invoked inside `body`, with the same effect as in the `for` command. `Foreach` returns an empty string.

EXAMPLES

The following loop uses `i` and `j` as loop variables to iterate over pairs of elements of a single list.

```
set x {}
foreach {i j} {a b c d e f} {
    lappend x $j $i
}
# The value of x is "b a d c f e"
# There are 3 iterations of the loop.
```

The next loop uses `i` and `j` to iterate over two lists in parallel.

```
set x {}
foreach i {a b c} j {d e f g} {
    lappend x $i $j
}
```

foreach

Tcl8.4/Tk8.4 Manual

116
# The value of x is "a d b e c f {} g"
# There are 4 iterations of the loop.

The two forms are combined in the following example.

```
set x {}
foreach i {a b c} {j k} {d e f g} {
    lappend x $i $j $k
}
# The value of x is "a d e b f g c {} {}"
# There are 3 iterations of the loop.
```

SEE ALSO

for, while, break, continue

KEYWORDS

foreach, iteration, list, looping

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
format

NAME

format – Format a string in the style of sprintf

SYNOPSIS

format formatString ?arg arg ...?

INTRODUCTION

DETAILS ON Formatting

- +
- space
- 0
- #
- d
- i
- o
- x or X
- e
- s
- f
- e or E
- g or G
- %

DIFFERENCES FROM ANSI SPRINTF

EXAMPLES

SEE ALSO

KEYWORDS
from format is the formatted string.

**DETAILS ON FORMATTING**

The command operates by scanning `formatString` from left to right. Each character from the format string is appended to the result string unless it is a percent sign. If the character is a % then it is not copied to the result string. Instead, the characters following the % character are treated as a conversion specifier. The conversion specifier controls the conversion of the next successive arg to a particular format and the result is appended to the result string in place of the conversion specifier. If there are multiple conversion specifiers in the format string, then each one controls the conversion of one additional arg. The format command must be given enough args to meet the needs of all of the conversion specifiers in `formatString`.

Each conversion specifier may contain up to six different parts: an XPG3 position specifier, a set of flags, a minimum field width, a precision, a length modifier, and a conversion character. Any of these fields may be omitted except for the conversion character. The fields that are present must appear in the order given above. The paragraphs below discuss each of these fields in turn.

If the % is followed by a decimal number and a $, as in ```%2$d```, then the value to convert is not taken from the next sequential argument. Instead, it is taken from the argument indicated by the number, where 1 corresponds to the first arg. If the conversion specifier requires multiple arguments because of * characters in the specifier then successive arguments are used, starting with the argument given by the number. This follows the XPG3 conventions for positional specifiers. If there are any positional specifiers in `formatString` then all of the specifiers must be positional.

The second portion of a conversion specifier may contain any of the following flag characters, in any order:

- `−` Specifies that the converted argument should be left–justified in its field (numbers are normally right–justified with leading spaces if needed).
- `+` Specifies that a number should always be printed with a sign, even if positive.
- `space` Specifies that a space should be added to the beginning of the number if the first character isn't a sign.
- `0` Specifies that the number should be padded on the left with zeroes instead of spaces.
- `#` Requests an alternate output form. For o and O conversions it guarantees that the first digit is always 0. For x or X conversions, 0x or 0X (respectively) will be added to the beginning of the result unless it is zero. For all floating–point conversions (e, E, f, g, and G) it guarantees that the result always has a decimal point. For g and G conversions it specifies that trailing zeroes should not be removed.

The third portion of a conversion specifier is a number giving a minimum field width for this conversion. It is typically used to make columns line up in tabular printouts. If the converted argument contains fewer characters than the minimum field width then it will be padded so that it is as wide as the minimum field width. Padding normally occurs by adding extra spaces on the left of the converted argument, but the 0 and –
flags may be used to specify padding with zeroes on the left or with spaces on the right, respectively. If the
minimum field width is specified as * rather than a number, then the next argument to the format command
determines the minimum field width; it must be a numeric string.

The fourth portion of a conversion specifier is a precision, which consists of a period followed by a number.
The number is used in different ways for different conversions. For e, E, and f conversions it specifies the
number of digits to appear to the right of the decimal point. For g and G conversions it specifies the total
number of digits to appear, including those on both sides of the decimal point (however, trailing zeroes after
the decimal point will still be omitted unless the # flag has been specified). For integer conversions, it
specifies a minimum number of digits to print (leading zeroes will be added if necessary). For s conversions it
specifies the maximum number of characters to be printed; if the string is longer than this then the trailing
characters will be dropped. If the precision is specified with * rather than a number then the next argument to
the format command determines the precision; it must be a numeric string.

The fifth part of a conversion specifier is a length modifier, which must be h or l. If it is h it specifies that the
numeric value should be truncated to a 16−bit value before converting. This option is rarely useful. If it is l it
specifies that the numeric value should be (at least) a 64−bit value. If neither h nor l are present, numeric
values are interpreted as being values of the width of the native machine word, as described by
tcl_platform(wordSize).

The last thing in a conversion specifier is an alphabetic character that determines what kind of conversion to
perform. The following conversion characters are currently supported:

dl
Convert integer to signed decimal string.

pu
Convert integer to unsigned decimal string.

ii
Convert integer to signed decimal string; the integer may either be in decimal, in octal (with a leading
0) or in hexadecimal (with a leading 0x).

oo
Convert integer to unsigned octal string.

xx or X
Convert integer to unsigned hexadecimal string, using digits `0123456789abcdef" for x and
"0123456789ABCDEF" for X).

c
c
Convert integer to the Unicode character it represents.

s
No conversion; just insert string.

ff
Convert floating−point number to signed decimal string of the form xx.yyy, where the number of y's is
determined by the precision (default: 6). If the precision is 0 then no decimal point is output.

ee or e
Convert floating−point number to scientific notation in the form x. yyyy ezz, where the number of y's is
determined by the precision (default: 6). If the precision is 0 then no decimal point is output. If the E
form is used then \texttt{E} is printed instead of \texttt{e}.

\textit{g} or \textit{G}

If the exponent is less than \(-4\) or greater than or equal to the precision, then convert floating-point number as for \texttt{%e} or \texttt{%E}. Otherwise convert as for \texttt{%f}. Trailing zeroes and a trailing decimal point are omitted.

\texttt{\%}

No conversion: just insert \texttt{\%}.

For the numerical conversions the argument being converted must be an integer or floating-point string; format converts the argument to binary and then converts it back to a string according to the conversion specifier.

**DIFFERENCES FROM ANSI SPRINTF**

The behavior of the format command is the same as the ANSI C \texttt{sprintf} procedure except for the following differences:

1. \texttt{\%p} and \texttt{\%n} specifiers are not currently supported.
2. For \texttt{\%c} conversions the argument must be a decimal string, which will then be converted to the corresponding character value.
3. The \texttt{l} modifier is ignored for real values and on 64-bit platforms, which are always converted as if the \texttt{l} modifier were present (i.e. the types \texttt{double} and \texttt{long} are used for the internal representation of real and integer values, respectively). If the \texttt{h} modifier is specified then integer values are truncated to \texttt{short} before conversion. Both \texttt{h} and \texttt{l} modifiers are ignored on all other conversions.

**EXAMPLES**

Convert the output of \texttt{time} into seconds to an accuracy of hundredths of a second:

```tcl
set us [lindex [time $someTclCode] 0]
puts [format "%.2f seconds to execute" [expr {$us / 1e6}]]
```

Create a packed X11 literal color specification:

```tcl
# Each color-component should be in range (0..255)
set color [format "#%02x%02x%02x" $r $g $b]
```

Use XPG3 format codes to allow reordering of fields (a technique that is often used in localized message catalogs; see \texttt{msgcat}) without reordering the data values passed to \texttt{format}:

```tcl
set fmt1 "Today, %d shares in %s were bought at $%.2f each"
puts [format $fmt1 123 "Global BigCorp" 19.37]

set fmt2 "Bought %2\$s equity ($%3$.2f x %1\$d) today"
```
puts [format $fmt2 123 "Global BigCorp" 19.37]

Print a small table of powers of three:

# Set up the column widths
set w1 5
set w2 10

# Make a nice header (with separator) for the table first
set sep +−[string repeat − $w1]−+−[string repeat − $w2]−+
puts $sep
puts [format "| %−*s | %−*s |" $w1 "Index" $w2 "Power"]
puts $sep

# Print the contents of the table
set p 1
for {set i 0} {$i<=20} {incr i} {
    puts [format "| %*d | %*ld |" $w1 $i $w2 $p]
    set p [expr {wide($p) * 3}]
}

# Finish off by printing the separator again
puts $sep

SEE ALSO

scan, sprintf, string

KEYWORDS

conversion specifier, format, sprintf, string, substitution

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
gets

NAME

gets – Read a line from a channel

SYNOPSIS

gets channelId ?varName?

DESCRIPTION

This command reads the next line from channelId, returns everything in the line up to (but not including) the end–of–line character(s), and discards the end–of–line character(s).

ChannelId must be an identifier for an open channel such as the Tcl standard input channel (stdin), the return value from an invocation of open or socket, or the result of a channel creation command provided by a Tcl extension. The channel must have been opened for input.

If varName is omitted the line is returned as the result of the command. If varName is specified then the line is placed in the variable by that name and the return value is a count of the number of characters returned.

If end of file occurs while scanning for an end of line, the command returns whatever input is available up to the end of file. If channelId is in nonblocking mode and there is not a full line of input available, the command returns an empty string and does not consume any input. If varName is specified and an empty string is returned in varName because of end–of–file or because of insufficient data in nonblocking mode, then the return count is −1. Note that if varName is not specified then the end–of–file and no–full–line–available cases can produce the same results as if there were an input line consisting only of the end–of–line character(s). The eof and fblocked commands can be used to distinguish these three cases.

EXAMPLE

This example reads a file one line at a time and prints it out with the current line number attached to the start of each line.

```tcl
set chan [open "some.file.txt"]
set lineNumber 0
while {[gets $chan line] >= 0} {  
    puts "{[incr lineNumber]}; $line"  
}
close $chan
```
SEE ALSO

file, eof, fblocked, Tcl_StandardChannels

KEYWORDS

blocking, channel, end of file, end of line, line, nonblocking, read

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME
glob – Return names of files that match patterns

SYNOPSIS

glob ?switches? pattern ?pattern ...?

DESCRIPTION

This command performs file name ``globbing'' in a fashion similar to the csh shell. It returns a list of the files whose names match any of the pattern arguments.

If the initial arguments to glob start with – then they are treated as switches. The following switches are currently supported:
−directory directory
Search for files which match the given patterns starting in the given directory. This allows searching of directories whose name contains glob−sensitive characters without the need to quote such characters explicitly. This option may not be used in conjunction with −path, which is used to allow searching for complete file paths whose names may contain glob−sensitive characters.

−join
The remaining pattern arguments are treated as a single pattern obtained by joining the arguments with directory separators.

−nocomplain
Allows an empty list to be returned without error; without this switch an error is returned if the result list would be empty.

−path pathPrefix
Search for files with the given pathPrefix where the rest of the name matches the given patterns. This allows searching for files with names similar to a given file (as opposed to a directory) even when the names contain glob−sensitive characters. This option may not be used in conjunction with −directory. For example, to find all files with the same root name as $path, but differing extensions, you should use glob −path [file rootname $path] .* which will work even if $path contains numerous glob−sensitive characters.

−tails
Only return the part of each file found which follows the last directory named in any −directory or −path path specification. Thus glob −tails −directory $dir * is equivalent to set pwd [pwd] ; cd $dir ; glob * ; cd $pwd. For −path specifications, the returned names will include the last path segment, so glob −tails −path [file rootname ~/foo.tex] .* will return paths like foo.aux foo.bib foo.tex etc.

−types typeList
Only list files or directories which match typeList, where the items in the list have two forms. The first form is like the −type option of the Unix find command: b (block special file), c (character special file), d (directory), f (plain file), l (symbolic link), p (named pipe), or s (socket), where multiple types may be specified in the list. Glob will return all files which match at least one of the types given. Note that symbolic links will be returned both if −types l is given, or if the target of a link matches the requested type. So, a link to a directory will be returned if −types d was specified.

The second form specifies types where all the types given must match. These are r, w, x as file permissions, and readonly, hidden as special permission cases. On the Macintosh, MacOS types and creators are also supported, where any item which is four characters long is assumed to be a MacOS type (e.g. TEXT). Items which are of the form {macintosh type XXXX} or {macintosh creator XXXX} will match types or creators respectively. Unrecognized types, or specifications of multiple MacOS types/creators will signal an error.

The two forms may be mixed, so −types {d f r w} will find all regular files OR directories that have both read AND write permissions. The following are equivalent:

    glob −type d *
    glob */

except that the first case doesn’t return the trailing ”/” and is more platform independent.
− −

Marks the end of switches. The argument following this one will be treated as a pattern even if it starts with a −.

The pattern arguments may contain any of the following special characters:

?                      Matches any single character.

*                      Matches any sequence of zero or more characters.

\[chars\]              Matches any single character in chars. If chars contains a sequence of the form a−b then any character between a and b (inclusive) will match.

\x                     Matches the character x.

\{a,b,...\}            Matches any of the strings a, b, etc.

On Unix, as with csh, a ``." at the beginning of a file's name or just after a ``/'' must be matched explicitly or with a { } construct, unless the ``−types hidden'' flag is given (since ``." at the beginning of a file's name indicates that it is hidden). On other platforms, files beginning with a ``." are handled no differently to any others, except the special directories ``." and ``..'' which must be matched explicitly (this is to avoid a recursive pattern like ``glob −join * * * *'' from recursing up the directory hierarchy as well as down). In addition, all ``/'' characters must be matched explicitly.

If the first character in a pattern is ``~'' then it refers to the home directory for the user whose name follows the ``~''. If the ``~'' is followed immediately by ``/'' then the value of the HOME environment variable is used.

The glob command differs from csh globbing in two ways. First, it does not sort its result list (use the lsort command if you want the list sorted). Second, glob only returns the names of files that actually exist; in csh no check for existence is made unless a pattern contains a ?, *, or [ ] construct.

When the glob command returns relative paths whose filenames start with a tilde ``~'' (for example through glob * or glob −tails, the returned list will not quote the tilde with ``./''). This means care must be taken if those names are later to be used with file join, to avoid them being interpreted as absolute paths pointing to a given user's home directory.

**PORTABILITY ISSUES**

Unlike other Tcl commands that will accept both network and native style names (see the filename manual entry for details on how native and network names are specified), the glob command only accepts native names.

**Windows**

For Windows UNC names, the servername and sharename components of the path may not contain ?,
*, or [] constructs. On Windows NT, if pattern is of the form `"~username@domain"` it refers to the home directory of the user whose account information resides on the specified NT domain server. Otherwise, user account information is obtained from the local computer. On Windows 95 and 98, glob accepts patterns like `".../"` and `"..../"` for successively higher up parent directories. Since the backslash character has a special meaning to the glob command, glob patterns containing Windows style path separators need special care. The pattern `C:\foo\*` is interpreted as `C:\foo\*` where \f will match the single character f and \* will match the single character * and will not be interpreted as a wildcard character. One solution to this problem is to use the Unix style forward slash as a path separator. Windows style paths can be converted to Unix style paths with the command `file join $path` (or `file normalize $path` in Tcl 8.4).

**Macintosh**

When using the options, `−directory`, `−join` or `−path`, glob assumes the directory separator for the entire pattern is the standard ```:``. When not using these options, glob examines each pattern argument and uses ```/``` unless the pattern contains a ```:``.

**EXAMPLES**

Find all the Tcl files in the current directory:

```bash
glob *.tcl
```

Find all the Tcl files in the user's home directory, irrespective of what the current directory is:

```bash
glob -directory ~ *.tcl
```

Find all subdirectories of the current directory:

```bash
glob -type d *
```

Find all files whose name contains an "a", a "b" or the sequence "cde":

```bash
glob -type f *{a,b,cde}*
```

**SEE ALSO**

`file`

**KEYWORDS**

exist, file, glob, pattern
global

NAME

global – Access global variables

SYNOPSIS

global varname ?varname ...?

DESCRIPTION

This command has no effect unless executed in the context of a proc body. If the global command is executed in the context of a proc body, it creates local variables linked to the corresponding global variables (and therefore these variables are listed by info locals).

If varname contains namespace qualifiers, the local variable's name is the unqualified name of the global variable, as determined by the namespace tail command.

EXAMPLES

This procedure sets the namespace variable ::a::x

    proc reset {} {
        global a::x
        set x 0
    }

This procedure accumulates the strings passed to it in a global buffer, separated by newlines. It is useful for situations when you want to build a message piece–by–piece (as if with puts) but send that full message in a single piece (e.g. over a connection opened with socket or as part of a counted HTTP response).

    proc accum {string} {
        global accumulator
        append accumulator $string \n
    }

SEE ALSO

namespace, upvar, variable

KEYWORDS

global, namespace, procedure, variable
NAME

history – Manipulate the history list

SYNOPSIS

history ?option? ?arg arg ...?

DESCRIPTION

The history command performs one of several operations related to recently-executed commands recorded in a history list. Each of these recorded commands is referred to as an "event". When specifying an event to the history command, the following forms may be used:

[1] A number: if positive, it refers to the event with that number (all events are numbered starting at 1). If the number is negative, it selects an event relative to the current event (−1 refers to the previous event, −2 to the one before that, and so on). Event 0 refers to the current event.

[2] A string: selects the most recent event that matches the string. An event is considered to match the string either if the string is the same as the first characters of the event, or if the string matches the event in the sense of the string match command.
The `history` command can take any of the following forms:

**history**
Same as `history info`, described below.

**history add command ?exec?**
Adds the `command` argument to the history list as a new event. If `exec` is specified (or abbreviated) then the command is also executed and its result is returned. If `exec` isn't specified then an empty string is returned as result.

**history change newValue ?event?**
Replaces the value recorded for an event with `newValue`. `Event` specifies the event to replace, and defaults to the current event (not event −1). This command is intended for use in commands that implement new forms of history substitution and wish to replace the current event (which invokes the substitution) with the command created through substitution. The return value is an empty string.

**history clear**
Erase the history list. The current keep limit is retained. The history event numbers are reset.

**history event ?event?**
Returns the value of the event given by `event`. `Event` defaults to −1.

**history info ?count?**
Returns a formatted string (intended for humans to read) giving the event number and contents for each of the events in the history list except the current event. If `count` is specified then only the most recent `count` events are returned.

**history keep ?count?**
This command may be used to change the size of the history list to `count` events. Initially, 20 events are retained in the history list. If `count` is not specified, the current keep limit is returned.

**history nextid**
Returns the number of the next event to be recorded in the history list. It is useful for things like printing the event number in command–line prompts.

**history redo ?event?**
Re–executes the command indicated by `event` and returns its result. `Event` defaults to −1. This command results in history revision: see below for details.

### HISTORY REVISION

Pre–8.0 Tcl had a complex history revision mechanism. The current mechanism is more limited, and the old history operations `substitute` and `words` have been removed. (As a consolation, the `clear` operation was added.)

The history option `redo` results in much simpler ”history revision”. When this option is invoked then the most recent event is modified to eliminate the history command and replace it with the result of the history command. If you want to redo an event without modifying history, then use the `event` operation to retrieve some event, and the `add` operation to add it to history and execute it.
NAME

http -- Client-side implementation of the HTTP/1.0 protocol.

SYNOPSIS

package require http ?2.5?
::http::config ?options?
::http::geturl url ?options?
::http::formatQuery key value ?key value ...?
::http::reset token ?why?
::http::wait token
::http::status token
::http::size token
::http::code token
::http::ncode token
::http::data token
::http::error token
::http::cleanup token
::http::register proto port command
::http::unregister proto

DESCRIPTION

COMMANDS

::http::config ?options?
   -accept mimetypes
   -proxyhost hostname
   -proxypart number
   -proxyfilter command
   -urlencoding encoding
   -useragent string
::http::geturl url ?options?
   -binary boolean
   -blocksize size
   -channel name
   -command callback
   -handler callback
   -headers keyvalue
   -progress callback
   -query query
   -queryblocksize size
   -querychannel channelID
   -queryprogress callback
   -timeout milliseconds
   -type mime-type
NAME

http – Client–side implementation of the HTTP/1.0 protocol.

SYNOPSIS

package require http ?2.5?
::http::config ?options?
::http::geturl url ?options?
::http::formatQuery key value ?key value ...?
::http::reset token ?why?
::http::wait token
::http::status token
::http::size token
::http::code token
::http::ncode token
::http::data token
::http::error token
::http::cleanup token
::http::register proto port command
::http:: unregister proto

DESCRIPTION

The http package provides the client side of the HTTP/1.0 protocol. The package implements the GET, POST, and HEAD operations of HTTP/1.0. It allows configuration of a proxy host to get through firewalls. The package is compatible with the Safesock security policy, so it can be used by untrusted applets to do URL fetching from a restricted set of hosts. This package can be extended to support additional HTTP transport protocols, such as HTTPS, by providing a custom socket command, via ::http::register.

The ::http::geturl procedure does a HTTP transaction. Its options determine whether a GET, POST, or HEAD transaction is performed. The return value of ::http::geturl is a token for the transaction. The value is also the name of an array in the ::http namespace that contains state information about the transaction. The elements of this array are described in the STATE ARRAY section.

If the --command option is specified, then the HTTP operation is done in the background. ::http::geturl returns immediately after generating the HTTP request and the callback is invoked when the transaction completes. For this to work, the Tcl event loop must be active. In Tk applications this is always true. For pure−Tcl applications, the caller can use ::http::wait after calling ::http::geturl to start the event loop.

COMMANDS

::http::config ?options?

The ::http::config command is used to set and query the name of the proxy server and port, and the User−Agent name used in the HTTP requests. If no options are specified, then the current configuration is returned. If a single argument is specified, then it should be one of the flags described below. In this case the current value of that setting is returned. Otherwise, the options should be a set of flags and values that define the configuration:

--accept mimetypes

The Accept header of the request. The default is */*, which means that all types of documents are accepted. Otherwise you can supply a comma−separated list of mime type patterns that you are willing to receive. For example, "image/gif, image/jpeg, text/*".
−proxyhost hostname
   The name of the proxy host, if any. If this value is the empty string, the URL host is contacted
directly.

−proxyport number
   The proxy port number.

−proxyfilter command
   The command is a callback that is made during ::http::geturl to determine if a proxy is
required for a given host. One argument, a host name, is added to command when it is
invoked. If a proxy is required, the callback should return a two−element list containing the
proxy server and proxy port. Otherwise the filter should return an empty list. The default filter
returns the values of the −proxyhost and −proxyport settings if they are non−empty.

−urlencoding encoding
   The encoding used for creating the x−url−encoded URLs with ::http::formatQuery. The
default is utf−8, as specified by RFC 2718. Prior to http 2.5 this was unspecified, and that
behavior can be returned by specifying the empty string ({}), although iso8859−1 is
recommended to restore similar behavior but without the ::http::formatQuery throwing an
error processing non−latin−1 characters.

−useragent string
   The value of the User−Agent header in the HTTP request. The default is "Tcl http client
package 2.4."

::http::geturl url ?options?
   The ::http::geturl command is the main procedure in the package. The −query option causes a
POST operation and the −validate option causes a HEAD operation; otherwise, a GET operation is
performed. The ::http::geturl command returns a token value that can be used to get information
about the transaction. See the STATE ARRAY and ERRORS section for details. The ::http::geturl
command blocks until the operation completes, unless the −command option specifies a callback that
is invoked when the HTTP transaction completes. ::http::geturl takes several options:

−binary boolean
   Specifies whether to force interpreting the URL data as binary. Normally this is auto−detected
(anything not beginning with a text content type or whose content encoding is gzip or
compress is considered binary data).

−blocksize size
   The block size used when reading the URL. At most size bytes are read at once. After each
block, a call to the −progress callback is made (if that option is specified).

−channel name
   Copy the URL contents to channel name instead of saving it in state(body).

−command callback
   Invoke callback after the HTTP transaction completes. This option causes ::http::geturl to
return immediately. The callback gets an additional argument that is the token returned from
::http::geturl. This token is the name of an array that is described in the STATE ARRAY
section. Here is a template for the callback:

proc httpCallback {token} {
  upvar #0 $token state
  # Access state as a Tcl array
–handler callback
Invoke callback whenever HTTP data is available; if present, nothing else will be done with the HTTP data. This procedure gets two additional arguments: the socket for the HTTP data and the token returned from ::http::geturl. The token is the name of a global array that is described in the STATE ARRAY section. The procedure is expected to return the number of bytes read from the socket. Here is a template for the callback:

```tcl
proc httpHandlerCallback {socket token} {
    upvar #0 $token state
    # Access socket, and state as a Tcl array
    ...
    (example: set data [read $socket 1000]; set nbytes [string length $data])
    ...
    return nbytes
}
```

–headers keyvaluelist
This option is used to add extra headers to the HTTP request. The keyvaluelist argument must be a list with an even number of elements that alternate between keys and values. The keys become header field names. Newlines are stripped from the values so the header cannot be corrupted. For example, if keyvaluelist is Pragma no–cache then the following header is included in the HTTP request:

Pragma: no–cache

–progress callback
The callback is made after each transfer of data from the URL. The callback gets three additional arguments: the token from ::http::geturl, the expected total size of the contents from the Content–Length meta–data, and the current number of bytes transferred so far. The expected total size may be unknown, in which case zero is passed to the callback. Here is a template for the progress callback:

```tcl
proc httpProgress {token total current} {
    upvar #0 $token state
}
```

–query query
This flag causes ::http::geturl to do a POST request that passes the query to the server. The query must be an x–url–encoding formatted query. The ::http::formatQuery procedure can be used to do the formatting.

–queryblocksize size
The block size used when posting query data to the URL. At most size bytes are written at once. After each block, a call to the –queryprogress callback is made (if that option is specified).

–querychannel channelID
This flag causes ::http::geturl to do a POST request that passes the data contained in channelID to the server. The data contained in channelID must be an x–url–encoding formatted query unless the –type option below is used. If a Content–Length header is not specified via the –headers options, ::http::geturl attempts to determine the size of the post data in order to create that header. If it is unable to determine the size, it returns an error.
The callback is made after each transfer of data to the URL (i.e. POST) and acts exactly like the –progress option (the callback format is the same).

–timeout milliseconds
If milliseconds is non-zero, then ::http::geturl sets up a timeout to occur after the specified number of milliseconds. A timeout results in a call to ::http::reset and to the –command callback, if specified. The return value of ::http::status is timeout after a timeout has occurred.

–type mime-type
Use mime-type as the Content-Type value, instead of the default value (application/x-www-form-urlencoded) during a POST operation.

–validate boolean
If boolean is non-zero, then ::http::geturl does an HTTP HEAD request. This request returns meta information about the URL, but the contents are not returned. The meta information is available in the state(meta) variable after the transaction. See the STATE ARRAY section for details.

::http::formatQuery key value ?key value ...?
This procedure does x-url-encoding of query data. It takes an even number of arguments that are the keys and values of the query. It encodes the keys and values, and generates one string that has the proper & and = separators. The result is suitable for the –query value passed to ::http::geturl.

::http::reset token ?why?
This command resets the HTTP transaction identified by token, if any. This sets the state(status) value to why, which defaults to reset, and then calls the registered –command callback.

::http::wait token
This is a convenience procedure that blocks and waits for the transaction to complete. This only works in trusted code because it uses vwait. Also, it’s not useful for the case where ::http::geturl is called without the –command option because in this case the ::http::geturl call doesn’t return until the HTTP transaction is complete, and thus there’s nothing to wait for.

::http::data token
This is a convenience procedure that returns the body element (i.e., the URL data) of the state array.

::http::error token
This is a convenience procedure that returns the error element of the state array.

::http::status token
This is a convenience procedure that returns the status element of the state array.

::http::code token
This is a convenience procedure that returns the http element of the state array.

::http::ncode token
This is a convenience procedure that returns just the numeric return code (200, 404, etc.) from the http element of the state array.

::http::size token
This is a convenience procedure that returns the currentsize element of the state array, which represents the number of bytes received from the URL in the ::http::geturl call.

::http::cleanup token
This procedure cleans up the state associated with the connection identified by token. After this call, the procedures like ::http::data cannot be used to get information about the operation. It is strongly
recommended that you call this function after you're done with a given HTTP request. Not doing so will result in memory not being freed, and if your app calls `::http::geturl` enough times, the memory leak could cause a performance hit...or worse.

`::http::register proto port command`

This procedure allows one to provide custom HTTP transport types such as HTTPS, by registering a prefix, the default port, and the command to execute to create the Tcl channel. E.g.:

```
package require http
package require tls

::http::register https 443 ::tls::socket

set token [::http::geturl https://my.secure.site/]
```

`::http::unregister proto`

This procedure unregisters a protocol handler that was previously registered via `::http::register`.

**ERRORS**

The `::http::geturl` procedure will raise errors in the following cases: invalid command line options, an invalid URL, a URL on a non-existent host, or a URL at a bad port on an existing host. These errors mean that it cannot even start the network transaction. It will also raise an error if it gets an I/O error while writing out the HTTP request header. For synchronous `::http::geturl` calls (where `−command` is not specified), it will raise an error if it gets an I/O error while reading the HTTP reply headers or data. Because `::http::geturl` doesn't return a token in these cases, it does all the required cleanup and there's no issue of your app having to call `::http::cleanup`.

For asynchronous `::http::geturl` calls, all of the above error situations apply, except that if there's any error while reading the HTTP reply headers or data, no exception is thrown. This is because after writing the HTTP headers, `::http::geturl` returns, and the rest of the HTTP transaction occurs in the background. The command callback can check if any error occurred during the read by calling `::http::status` to check the status and if its `error`, calling `::http::error` to get the error message.

Alternatively, if the main program flow reaches a point where it needs to know the result of the asynchronous HTTP request, it can call `::http::wait` and then check status and error, just as the callback does.

In any case, you must still call `::http::cleanup` to delete the state array when you're done.

There are other possible results of the HTTP transaction determined by examining the status from `::http::status`. These are described below.

* `ok`  
  If the HTTP transaction completes entirely, then status will be `ok`. However, you should still check the `::http::code` value to get the HTTP status. The `::http::ncode` procedure provides just the numeric error (e.g., 200, 404 or 500) while the `::http::code` procedure returns a value like "HTTP 404 File not found".

* `eof`
If the server closes the socket without replying, then no error is raised, but the status of the transaction will be `eof`.

**error**

The error message will also be stored in the `error` status array element, accessible via `::http::error`.

Another error possibility is that `::http::geturl` is unable to write all the post query data to the server before the server responds and closes the socket. The error message is saved in the `posterror` status array element and then `::http::geturl` attempts to complete the transaction. If it can read the server's response it will end up with an `ok` status, otherwise it will have an `eof` status.

**STATE ARRAY**

The `::http::geturl` procedure returns a `token` that can be used to get to the state of the HTTP transaction in the form of a Tcl array. Use this construct to create an easy-to-use array variable:

```
upvar #0 $token state
```

Once the data associated with the URL is no longer needed, the state array should be unset to free up storage. The `::http::cleanup` procedure is provided for that purpose. The following elements of the array are supported:

**body**

The contents of the URL. This will be empty if the `-channel` option has been specified. This value is returned by the `::http::data` command.

**charset**

The value of the charset attribute from the `Content-Type` meta-data value. If none was specified, this defaults to the RFC standard `iso8859-1`, or the value of `::http::defaultCharset`. Incoming text data will be automatically converted from this charset to utf-8.

**coding**

A copy of the `Content-Encoding` meta-data value.

**currentsize**

The current number of bytes fetched from the URL. This value is returned by the `::http::size` command.

**error**

If defined, this is the error string seen when the HTTP transaction was aborted.

**http**

The HTTP status reply from the server. This value is returned by the `::http::code` command. The format of this value is:

```
HTTP/1.0 code string
```

The `code` is a three-digit number defined in the HTTP standard. A code of 200 is OK. Codes beginning with 4 or 5 indicate errors. Codes beginning with 3 are redirection errors. In this case the Location meta-data specifies a new URL that contains the requested information.

**meta**
The HTTP protocol returns meta-data that describes the URL contents. The `meta` element of the state array is a list of the keys and values of the meta-data. This is in a format useful for initializing an array that just contains the meta-data:

```
array set meta $state(meta)
```

Some of the meta-data keys are listed below, but the HTTP standard defines more, and servers are free to add their own.

**Content-Type**

The type of the URL contents. Examples include `text/html`, `image/gif`, `application/postscript` and `application/x-tcl`.

**Content-Length**

The advertised size of the contents. The actual size obtained by `::http::geturl` is available as `state(size)`.

**Location**

An alternate URL that contains the requested data.

**posterror**

The error, if any, that occurred while writing the post query data to the server.

**status**

Either `ok`, for successful completion, `reset` for user-reset, `timeout` if a timeout occurred before the transaction could complete, or `error` for an error condition. During the transaction this value is the empty string.

**totalsize**

A copy of the `Content-Length` meta-data value.

**type**

A copy of the `Content-Type` meta-data value.

**url**

The requested URL.

**EXAMPLE**

```
# Copy a URL to a file and print meta-data
proc httpcopy { url file {chunk 4096} } {
    set out [open $file w]
    set token [::http::geturl $url -channel $out \    
               -progress httpCopyProgress -blocksize $chunk]
    close $out

    # This ends the line started by httpCopyProgress
    puts stderr ""
    upvar #0 $token state
    set max 0
    foreach {name value} $state(meta) {
        if {[string length $name] > $max} {
            set max [string length $name]
        }
    }
```
if {{regexp −nocase ^location$ $name} {  
    # Handle URL redirects
    puts stderr "Location:$value"
    return [httpcopy [string trim $value] $file $chunk]
    }
}
incr max
foreach {name value} $state(meta) {
    puts [format "%-*s %s $max $name: $value"

    return $token
}
proc httpCopyProgress {args} {
    puts −nonewline stderr .
    flush stderr
}

SEE ALSO

safe, socket, safesock

KEYWORDS

security policy, socket

Copyright ' 1995-1997 Sun Microsystems, Inc.
Copyright ' 1998-2000 by Ajuba Solutions.
Copyright ' 2004 ActiveState Corporation.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
if

NAME

if – Execute scripts conditionally

SYNOPSIS

if expr1 ?then? body1 elseif expr2 ?then? body2 elseif ... ?else? ?bodyN?

DESCRIPTION

The if command evaluates expr1 as an expression (in the same way that expr evaluates its argument). The value of the expression must be a boolean (a numeric value, where 0 is false and anything is true, or a string value such as true or yes for true and false or no for false); if it is true then body1 is executed by passing it to the Tcl interpreter. Otherwise expr2 is evaluated as an expression and if it is true then body2 is executed, and so on. If none of the expressions evaluates to true then bodyN is executed. The then and else arguments are optional "noise words" to make the command easier to read. There may be any number of elseif clauses, including zero. BodyN may also be omitted as long as else is omitted too. The return value from the command is the result of the body script that was executed, or an empty string if none of the expressions was non-zero and there was no bodyN.

EXAMPLES

A simple conditional:

if {$vbl == 1} { puts "vbl is one" }

With an else–clause:

if {$vbl == 1} {
    puts "vbl is one"
} else {
    puts "vbl is not one"
}

With an elseif–clause too:

if {$vbl == 1} {
    puts "vbl is one"
} elseif {$vbl == 2} {
    puts "vbl is two"
} else {
    puts "vbl is not one or two"
}
Remember, expressions can be multi-line, but in that case it can be a good idea to use the optional then keyword for clarity:

```tcl
if {
    $vbl == 1 || $vbl == 2 || $vbl == 3
} then {
    puts "vbl is one, two or three"
}
```

SEE ALSO

`expr`, `for`, `foreach`

KEYWORDS

`boolean, conditional, else, false, if, true`

Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
Copyright © 1995-1997 Roger E. Critchlow Jr.
incr

NAME

incr – Increment the value of a variable

SYNOPSIS

incr varName ?increment?

DESCRIPTION

Increments the value stored in the variable whose name is varName. The value of the variable must be an integer. If increment is supplied then its value (which must be an integer) is added to the value of variable varName; otherwise 1 is added to varName. The new value is stored as a decimal string in variable varName and also returned as result.

EXAMPLES

Add one to the contents of the variable x:

incr x

Add 42 to the contents of the variable x:

incr x 42

Add the contents of the variable y to the contents of the variable x:

incr x $y

Add nothing at all to the variable x (often useful for checking whether an argument to a procedure is actually numeric and generating an error if it is not):

incr x 0

SEE ALSO

expr

KEYWORDS

add, increment, variable, value
NAME

info – Return information about the state of the Tcl interpreter

SYNOPSIS

info option ?arg arg ...?

DESCRIPTION

info args procname
info body procname
info cmdcount
info commands ?pattern?
info complete command
info default procname arg varname
info exists varName
info functions ?pattern?
info globals ?pattern?
info hostname
info level ?number?
info library
info loaded ?interp?
info locals ?pattern?
info nameofexecutable
info patchlevel
info procs ?pattern?
info script ?filename?
info sharedlibextension
info tclversion
info vars ?pattern?

EXAMPLE

SEE ALSO

KEYWORDS
DESCRIPTION

This command provides information about various internals of the Tcl interpreter. The legal option's (which may be abbreviated) are:

`info args procname`
Returns a list containing the names of the arguments to procedure `procname`, in order. `Procname` must be the name of a Tcl command procedure.

`info body procname`
Returns the body of procedure `procname`. `Procname` must be the name of a Tcl command procedure.

`info cmdcount`?
Returns a count of the total number of commands that have been invoked in this interpreter.

`info commands ?pattern?`?
If `pattern` isn't specified, returns a list of names of all the Tcl commands in the current namespace, including both the built-in commands written in C and the command procedures defined using the `proc` command. If `pattern` is specified, only those names matching `pattern` are returned. Matching is determined using the same rules as for string match. `Pattern` can be a qualified name like `Foo::print*`. That is, it may specify a particular namespace using a sequence of namespace names separated by double colons (`::`), and may have pattern matching special characters at the end to specify a set of commands in that namespace. If `pattern` is a qualified name, the resulting list of command names has each one qualified with the name of the specified namespace.

`info complete command`?
Returns 1 if `command` is a complete Tcl command in the sense of having no unclosed quotes, braces, brackets or array element names. If the command doesn't appear to be complete then 0 is returned. This command is typically used in line-oriented input environments to allow users to type in commands that span multiple lines; if the command isn't complete, the script can delay evaluating it until additional lines have been typed to complete the command.

`info default procname arg varname`?
`Procname` must be the name of a Tcl command procedure and `arg` must be the name of an argument to that procedure. If `arg` doesn't have a default value then the command returns 0. Otherwise it returns 1 and places the default value of `arg` into variable `varname`.

`info exists varName`?
Returns 1 if the variable named `varName` exists in the current context (either as a global or local variable) and has been defined by being given a value, returns 0 otherwise.

`info functions ?pattern?`?
If `pattern` isn't specified, returns a list of all the math functions currently defined. If `pattern` is specified, only those functions whose name matches `pattern` are returned. Matching is determined using the same rules as for string match.

`info globals ?pattern?`?
If `pattern` isn't specified, returns a list of all the names of currently-defined global variables. Global variables are variables in the global namespace. If `pattern` is specified, only those names matching `pattern` are returned. Matching is determined using the same rules as for string match.

`info hostname`?
Returns the name of the computer on which this invocation is being executed. Note that this name is not guaranteed to be the fully qualified domain name of the host. Where machines have several
different names (as is common on systems with both TCP/IP (DNS) and NetBIOS–based networking installed,) it is the name that is suitable for TCP/IP networking that is returned.

**info level **?number?

If `number` is not specified, this command returns a number giving the stack level of the invoking procedure, or 0 if the command is invoked at top–level. If `number` is specified, then the result is a list consisting of the name and arguments for the procedure call at level `number` on the stack. If `number` is positive then it selects a particular stack level (1 refers to the top–most active procedure, 2 to the procedure it called, and so on); otherwise it gives a level relative to the current level (0 refers to the current procedure, −1 to its caller, and so on). See the `uplevel` command for more information on what stack levels mean.

**info library**

Returns the name of the library directory in which standard Tcl scripts are stored. This is actually the value of the `tcl_library` variable and may be changed by setting `tcl_library`. See the `tclvars` manual entry for more information.

**info loaded **? interp?  

Returns a list describing all of the packages that have been loaded into `interp` with the `load` command. Each list element is a sub–list with two elements consisting of the name of the file from which the package was loaded and the name of the package. For statically–loaded packages the file name will be an empty string. If `interp` is omitted then information is returned for all packages loaded in any interpreter in the process. To get a list of just the packages in the current interpreter, specify an empty string for the `interp` argument.

**info locals **?pattern?

If `pattern` isn't specified, returns a list of all the names of currently–defined local variables, including arguments to the current procedure, if any. Variables defined with the `global`, `upvar` and `variable` commands will not be returned. If `pattern` is specified, only those names matching `pattern` are returned. Matching is determined using the same rules as for `string match`.

**info nameofexecutable**

Returns the full path name of the binary file from which the application was invoked. If Tcl was unable to identify the file, then an empty string is returned.

**info patchlevel**

Returns the value of the global variable `tcl_patchLevel`; see the `tclvars` manual entry for more information.

**info procs **?pattern?

If `pattern` isn't specified, returns a list of all the names of Tcl command procedures in the current namespace. If `pattern` is specified, only those procedure names in the current namespace matching `pattern` are returned. Matching is determined using the same rules as for `string match`. If `pattern` contains any namespace separators, they are used to select a namespace relative to the current namespace (or relative to the global namespace if `pattern` starts with `::`) to match within; the matching pattern is taken to be the part after the last namespace separator.

**info script **?filename?

If a Tcl script file is currently being evaluated (i.e. there is a call to `Tcl_EvalFile` active or there is an active invocation of the `source` command), then this command returns the name of the innermost file being processed. If `filename` is specified, then the return value of this command will be modified for the duration of the active invocation to return that name. This is useful in virtual file system applications. Otherwise the command returns an empty string.
info sharedlibextension

Returns the extension used on this platform for the names of files containing shared libraries (for example, .so under Solaris). If shared libraries aren't supported on this platform then an empty string is returned.

info tclversion

Returns the value of the global variable tcl_version; see the tclvars manual entry for more information.

info vars ?pattern?

If pattern isn't specified, returns a list of all the names of currently-visible variables. This includes locals and currently-visible globals. If pattern is specified, only those names matching pattern are returned. Matching is determined using the same rules as for string match, pattern can be a qualified name like Foo::option*. That is, it may specify a particular namespace using a sequence of namespace names separated by double colons (::), and may have pattern matching special characters at the end to specify a set of variables in that namespace. If pattern is a qualified name, the resulting list of variable names has each matching namespace variable qualified with the name of its namespace. Note that a currently-visible variable may not yet "exist" if it has not been set (e.g. a variable declared but not set by variable).

EXAMPLE

This command prints out a procedure suitable for saving in a Tcl script:

```tcl
proc printProc {procName} {
    set result [list proc $procName]
    set formals {}
    foreach var [info args $procName] {
        if {[info default $procName $var def]} {
            lappend formals [list $var $def]
        } else {
            # Still need the list-quoting because variable
            # names may properly contain spaces.
            lappend formals [list $var]
        }
    }
    puts [lappend result $formals [info body $procName]]
}
```

SEE ALSO

global, proc

KEYWORDS

cmd, info, options, proc, variable

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1997 Sun Microsystems, Inc.
Copyright ' 1993-1997 Bell Labs Innovations for Lucent Technologies
NAME

interp – Create and manipulate Tcl interpreters

SYNOPSIS

interp option ?arg arg ...?

DESCRIPTION

THE INTERP COMMAND

interp alias srcPath srcToken
interp alias srcPath srcToken {}
interp alias srcPath srcCmd targetPath targetCmd ?arg arg ...?
interp aliases ?path?
interp create ?−safe? ?−− ?path?
interp delete ?path ...?
interp eval path arg ?arg ...?
interp exists path
interp expose path hiddenName ?exposedCmdName?
interp hide path exposedCmdName ?hiddenCmdName?
interp hidden path
interp invokehidden path ?−global? hiddenCmdName ?arg ...?
interp issafe ?path?
interp marktrusted path
interp recursionlimit path ?newlimit?
interp share srcPath channelId destPath
interp slaves ?path?
interp target path alias
interp transfer srcPath channelId destPath

SLAVE COMMAND

slave aliases
slave alias srcToken
slave alias srcToken {}
slave alias srcCmd targetCmd ?arg ..?
slave eval arg ?arg ..?
slave expose hiddenName ?exposedCmdName?
slave hide exposedCmdName ?hiddenCmdName?
slave hidden
slave invokehidden ?−global hiddenName ?arg ..?
slave issafe
slave marktrusted
slave recursionlimit ?newlimit?

SAFE INTERPRETERS

ALIAS INVOCATION

HIDDEN COMMANDS
NAME

interp – Create and manipulate Tcl interpreters

SYNOPSIS

interp option ?arg arg ...?

DESCRIPTION

This command makes it possible to create one or more new Tcl interpreters that co–exist with the creating interpreter in the same application. The creating interpreter is called the master and the new interpreter is called a slave. A master can create any number of slaves, and each slave can itself create additional slaves for which it is master, resulting in a hierarchy of interpreters.

Each interpreter is independent from the others: it has its own name space for commands, procedures, and global variables. A master interpreter may create connections between its slaves and itself using a mechanism called an alias. An alias is a command in a slave interpreter which, when invoked, causes a command to be invoked in its master interpreter or in another slave interpreter. The only other connections between interpreters are through environment variables (the env variable), which are normally shared among all interpreters in the application. Note that the name space for files (such as the names returned by the open command) is no longer shared between interpreters. Explicit commands are provided to share files and to transfer references to open files from one interpreter to another.

The interp command also provides support for safe interpreters. A safe interpreter is a slave whose functions have been greatly restricted, so that it is safe to execute untrusted scripts without fear of them damaging other interpreters or the application's environment. For example, all IO channel creation commands and subprocess creation commands are made inaccessible to safe interpreters. See SAFE INTERPRETERS below for more information on what features are present in a safe interpreter. The dangerous functionality is not removed from the safe interpreter; instead, it is hidden, so that only trusted interpreters can obtain access to it. For a detailed explanation of hidden commands, see HIDDEN COMMANDS, below. The alias mechanism can be used for protected communication (analogous to a kernel call) between a slave interpreter and its master. See ALIAS INVOCATION, below, for more details on how the alias mechanism works.

A qualified interpreter name is a proper Tcl lists containing a subset of its ancestors in the interpreter hierarchy, terminated by the string naming the interpreter in its immediate master. Interpreter names are relative to the interpreter in which they are used. For example, if a is a slave of the current interpreter and it has a slave a1, which in turn has a slave a11, the qualified name of a11 in a is the list a1 a11.
The `interp` command, described below, accepts qualified interpreter names as arguments; the interpreter in which the command is being evaluated can always be referred to as `{}` (the empty list or string). Note that it is impossible to refer to a master (ancestor) interpreter by name in a slave interpreter except through aliases. Also, there is no global name by which one can refer to the first interpreter created in an application. Both restrictions are motivated by safety concerns.

**THE interp COMMAND**

The `interp` command is used to create, delete, and manipulate slave interpreters, and to share or transfer channels between interpreters. It can have any of several forms, depending on the `option` argument:

```tcl
interp alias srcPath srcToken
interp alias srcPath srcToken {}
interp alias srcPath srcCmd targetPath targetCmd ?arg arg ...?
interp aliases ?path?
interp create ?−safe? ?− − − ?path?
```

- **interp alias srcPath srcToken**
  Returns a Tcl list whose elements are the `targetCmd` and `args` associated with the alias represented by `srcToken` (this is the value returned when the alias was created; it is possible that the name of the source command in the slave is different from `srcToken`).

- **interp alias srcPath srcToken `{}`**
  Deletes the alias for `srcToken` in the slave interpreter identified by `srcPath`. `srcToken` refers to the value returned when the alias was created; if the source command has been renamed, the renamed command will be deleted.

- **interp alias srcPath srcCmd targetPath targetCmd ?arg arg ...?**
  This command creates an alias between one slave and another (see the `alias` slave command below for creating aliases between a slave and its master). In this command, either of the slave interpreters may be anywhere in the hierarchy of interpreters under the interpreter invoking the command. `SrcPath` and `srcCmd` identify the source of the alias. `SrcPath` is a Tcl list whose elements select a particular interpreter. For example, ``a b`` identifies an interpreter `b`, which is a slave of interpreter `a`, which is a slave of the invoking interpreter. An empty list specifies the interpreter invoking the command. `srcCmd` gives the name of a new command, which will be created in the source interpreter. `TargetPath` and `targetCmd` specify a target interpreter and command, and the `arg` arguments, if any, specify additional arguments to `targetCmd` which are prepended to any arguments specified in the invocation of `srcCmd`. `TargetCmd` may be undefined at the time of this call, or it may already exist; it is not created by this command. The alias arranges for the given target command to be invoked in the target interpreter whenever the given source command is invoked in the source interpreter. See **ALIAS INVOCATION** below for more details. The command returns a token that uniquely identifies the command created `srcCmd`, even if the command is renamed afterwards. The token may but does not have to be equal to `srcCmd`.

- **interp aliases ?path?**
  This command returns a Tcl list of the tokens of all the source commands for aliases defined in the interpreter identified by `path`. The tokens correspond to the values returned when the aliases were created (which may not be the same as the current names of the commands).

- **interp create ?−safe? ?− − − ?path?**
  Creates a slave interpreter identified by `path` and a new command, called a slave command. The name of the slave command is the last component of `path`. The new slave interpreter and the slave command are created in the interpreter identified by the path obtained by removing the last component from `path`. For example, if `path` is `a b c` then a new slave interpreter and slave command named `c` are created in the interpreter identified by the path `a b`. The slave command may be used to manipulate
the new interpreter as described below. If *path* is omitted, Tcl creates a unique name of the form `interp`, where *x* is an integer, and uses it for the interpreter and the slave command. If the `−safe` switch is specified (or if the master interpreter is a safe interpreter), the new slave interpreter will be created as a safe interpreter with limited functionality; otherwise the slave will include the full set of Tcl built-in commands and variables. The `− −` switch can be used to mark the end of switches; it may be needed if *path* is an unusual value such as `−safe`. The result of the command is the name of the new interpreter. The name of a slave interpreter must be unique among all the slaves for its master; an error occurs if a slave interpreter by the given name already exists in this master. The initial recursion limit of the slave interpreter is set to the current recursion limit of its parent interpreter.

**interp delete ?*path* ...?**

Deletes zero or more interpreters given by the optional *path* arguments, and for each interpreter, it also deletes its slaves. The command also deletes the slave command for each interpreter deleted. For each *path* argument, if no interpreter by that name exists, the command raises an error.

**interp eval *path* *arg* ?*arg* ...?**

This command concatenates all of the *arg* arguments in the same fashion as the `concat` command, then evaluates the resulting string as a Tcl script in the slave interpreter identified by *path*. The result of this evaluation (including error information such as the `errorInfo` and `errorCode` variables, if an error occurs) is returned to the invoking interpreter. Note that the script will be executed in the current context stack frame of the *path* interpreter; this is so that the implementations (in a master interpreter) of aliases in a slave interpreter can execute scripts in the slave that find out information about the slave's current state and stack frame.

**interp exists *path***

Returns 1 if a slave interpreter by the specified *path* exists in this master, 0 otherwise. If *path* is omitted, the invoking interpreter is used.

**interp expose *path* *hiddenName* ?*exposedCmdName*?**

Makes the hidden command *hiddenName* exposed, eventually bringing it back under a new *exposedCmdName* name (this name is currently accepted only if it is a valid global name space name without any ::), in the interpreter denoted by *path*. If an exposed command with the targeted name already exists, this command fails. Hidden commands are explained in more detail in [HIDDEN COMMANDS](#).

**interp hide *path* *exposedCmdName* ?*hiddenCmdName*?**

Makes the exposed command *exposedCmdName* hidden, renaming it to the hidden command *hiddenCmdName*, or keeping the same name if *hiddenCmdName* is not given, in the interpreter denoted by *path*. If a hidden command with the targeted name already exists, this command fails. Currently both *exposedCmdName* and *hiddenCmdName* can not contain namespace qualifiers, or an error is raised. Commands to be hidden by `interp hide` are looked up in the global namespace even if the current namespace is not the global one. This prevents slaves from fooling a master interpreter into hiding the wrong command, by making the current namespace be different from the global one. Hidden commands are explained in more detail in [HIDDEN COMMANDS](#).

**interp hidden *path***

Returns a list of the names of all hidden commands in the interpreter identified by *path*.

**interp invokehidden *path* ?−global? *hiddenCmdName* ?*arg* ...?**

Invokes the hidden command *hiddenCmdName* with the arguments supplied in the interpreter denoted by *path*. No substitutions or evaluation are applied to the arguments. If the `−global` flag is present, the hidden command is invoked at the global level in the target interpreter; otherwise it is invoked at the
current call frame and can access local variables in that and outer call frames. Hidden commands are explained in more detail in **HIDDEN COMMANDS**, below.

**interp issafe ?path?**

Returns 1 if the interpreter identified by the specified path is safe, 0 otherwise.

**interp marktrusted path**

Marks the interpreter identified by path as trusted. Does not expose the hidden commands. This command can only be invoked from a trusted interpreter. The command has no effect if the interpreter identified by path is already trusted.

**interp recursionlimit path ?newlimit?**

Returns the maximum allowable nesting depth for the interpreter specified by path. If newlimit is specified, the interpreter recursion limit will be set so that nesting of more than newlimit calls to Tcl_Eval() and related procedures in that interpreter will return an error. The newlimit value is also returned. The newlimit value must be a positive integer between 1 and the maximum value of a non–long integer on the platform.

The command sets the maximum size of the Tcl call stack only. It cannot by itself prevent stack overflows on the C stack being used by the application. If your machine has a limit on the size of the C stack, you may get stack overflows before reaching the limit set by the command. If this happens, see if there is a mechanism in your system for increasing the maximum size of the C stack.

**interp share srcPath channelId destPath**

Causes the IO channel identified by channelId to become shared between the interpreter identified by srcPath and the interpreter identified by destPath. Both interpreters have the same permissions on the IO channel. Both interpreters must close it to close the underlying IO channel; IO channels accessible in an interpreter are automatically closed when an interpreter is destroyed.

**interp slaves ?path?**

Returns a Tcl list of the names of all the slave interpreters associated with the interpreter identified by path. If path is omitted, the invoking interpreter is used.

**interp target path alias**

Returns a Tcl list describing the target interpreter for an alias. The alias is specified with an interpreter path and source command name, just as in **interp alias** above. The name of the target interpreter is returned as an interpreter path, relative to the invoking interpreter. If the target interpreter for the alias is the invoking interpreter then an empty list is returned. If the target interpreter for the alias is not the invoking interpreter or one of its descendants then an error is generated. The target command does not have to be defined at the time of this invocation.

**interp transfer srcPath channelId destPath**

Causes the IO channel identified by channelId to become available in the interpreter identified by destPath and unavailable in the interpreter identified by srcPath.

**SLAVE COMMAND**

For each slave interpreter created with the **interp** command, a new Tcl command is created in the master interpreter with the same name as the new interpreter. This command may be used to invoke various operations on the interpreter. It has the following general form:

```
slave command ?arg arg ...?
```
Slave is the name of the interpreter, and command and the args determine the exact behavior of the command. The valid forms of this command are:

slave aliases
Returns a Tcl list whose elements are the tokens of all the aliases in slave. The tokens correspond to the values returned when the aliases were created (which may not be the same as the current names of the commands).

slave alias srcToken
Returns a Tcl list whose elements are the targetCmd and args associated with the alias represented by srcToken (this is the value returned when the alias was created; it is possible that the actual source command in the slave is different from srcToken).

slave alias srcToken {}
Deletes the alias for srcToken in the slave interpreter. srcToken refers to the value returned when the alias was created; if the source command has been renamed, the renamed command will be deleted.

slave alias srcCmd targetCmd ?arg ..?
Creates an alias such that whenever srcCmd is invoked in slave, targetCmd is invoked in the master. The arg arguments will be passed to targetCmd as additional arguments, prepended before any arguments passed in the invocation of srcCmd. See ALIAS INVOCATION below for details. The command returns a token that uniquely identifies the command created srcCmd, even if the command is renamed afterwards. The token may but does not have to be equal to srcCmd.

slave eval arg ?arg ..?
This command concatenates all of the arg arguments in the same fashion as the concat command, then evaluates the resulting string as a Tcl script in slave. The result of this evaluation (including error information such as the errorInfo and errorCode variables, if an error occurs) is returned to the invoking interpreter. Note that the script will be executed in the current context stack frame of slave; this is so that the implementations (in a master interpreter) of aliases in a slave interpreter can execute scripts in the slave that find out information about the slave's current state and stack frame.

slave expose hiddenName ?exposedCmdName?
This command exposes the hidden command hiddenName, eventually bringing it back under a new exposedCmdName name (this name is currently accepted only if it is a valid global name space name without any ::), in slave. If an exposed command with the targeted name already exists, this command fails. For more details on hidden commands, see HIDDEN COMMANDS, below.

slave hide exposedCmdName ?hiddenCmdName?
This command hides the exposed command exposedCmdName, renaming it to the hidden command hiddenCmdName, or keeping the same name if the argument is not given, in the slave interpreter. If a hidden command with the targeted name already exists, this command fails. Currently both exposedCmdName and hiddenCmdName can not contain namespace qualifiers, or an error is raised. Commands to be hidden are looked up in the global namespace even if the current namespace is not the global one. This prevents slaves from fooling a master interpreter into hiding the wrong command, by making the current namespace be different from the global one. For more details on hidden commands, see HIDDEN COMMANDS, below.

slave hidden
Returns a list of the names of all hidden commands in slave.

slave invokehidden ?−global hiddenName ?arg ..?
This command invokes the hidden command hiddenName with the supplied arguments, in slave. No
substitutions or evaluations are applied to the arguments. If the \texttt{--global} flag is given, the command is invoked at the global level in the slave; otherwise it is invoked at the current call frame and can access local variables in that or outer call frames. For more details on hidden commands, see \texttt{HIDDEN COMMANDS}, below.

\texttt{slave issafe}

Returns 1 if the slave interpreter is safe, 0 otherwise.

\texttt{slave marktrusted}

Marks the slave interpreter as trusted. Can only be invoked by a trusted interpreter. This command does not expose any hidden commands in the slave interpreter. The command has no effect if the slave is already trusted.

\texttt{slave recursionlimit ?newlimit?}

Returns the maximum allowable nesting depth for the slave interpreter. If \texttt{newlimit} is specified, the recursion limit in \texttt{slave} will be set so that nesting of more than \texttt{newlimit} calls to \texttt{Tcl_Eval()} and related procedures in \texttt{slave} will return an error. The \texttt{newlimit} value is also returned. The \texttt{newlimit} value must be a positive integer between 1 and the maximum value of a non–long integer on the platform.

The command sets the maximum size of the Tcl call stack only. It cannot by itself prevent stack overflows on the C stack being used by the application. If your machine has a limit on the size of the C stack, you may get stack overflows before reaching the limit set by the command. If this happens, see if there is a mechanism in your system for increasing the maximum size of the C stack.

\section*{SAFE INTERPRETERS}

A safe interpreter is one with restricted functionality, so that is safe to execute an arbitrary script from your worst enemy without fear of that script damaging the enclosing application or the rest of your computing environment. In order to make an interpreter safe, certain commands and variables are removed from the interpreter. For example, commands to create files on disk are removed, and the \texttt{exec} command is removed, since it could be used to cause damage through subprocesses. Limited access to these facilities can be provided, by creating aliases to the master interpreter which check their arguments carefully and provide restricted access to a safe subset of facilities. For example, file creation might be allowed in a particular subdirectory and subprocess invocation might be allowed for a carefully selected and fixed set of programs.

A safe interpreter is created by specifying the \texttt{--safe} switch to the \texttt{interp create} command. Furthermore, any slave created by a safe interpreter will also be safe.

A safe interpreter is created with exactly the following set of built–in commands:
The following commands are hidden by `interp create` when it creates a safe interpreter:

```
cd encoding exec exit
fconfigure file glob load
open pwd socket source
```

These commands can be recreated later as Tcl procedures or aliases, or re-exposed by `interp expose`.

The following commands from Tcl's library of support procedures are not present in a safe interpreter:

```
auto_exec_ok auto_import auto_load
auto_load_index auto_qualify unknown
```

Note in particular that safe interpreters have no default `unknown` command, so Tcl's default autoloading facilities are not available. Autoload access to Tcl's commands that are normally autoloading:

```
auto_mkindex auto_mkindex_old
auto_reset history
parray pkg_mkIndex ::pkg::create ::safe::interpAddToAccessPath
::safe::interpCreate ::safe::interpConfigure
::safe::interpDelete ::safe::interpFindInAccessPath
::safe::interpInit ::safe::setLogCmd
```

...can only be provided by explicit definition of an `unknown` command in the safe interpreter. This will involve exposing the `source` command. This is most easily accomplished by creating the safe interpreter with Tcl's Safe−Tcl mechanism. Safe−Tcl provides safe versions of `source`, `load`, and other Tcl commands needed to support autoloading of commands and the loading of packages.

In addition, the `env` variable is not present in a safe interpreter, so it cannot share environment variables with other interpreters. The `env` variable poses a security risk, because users can store sensitive information in an environment variable. For example, the PGP manual recommends storing the PGP private key protection password in the environment variable `PGPPASS`. Making this variable available to untrusted code executing in a safe interpreter would incur a security risk.

If extensions are loaded into a safe interpreter, they may also restrict their own functionality to eliminate unsafe commands. For a discussion of management of extensions for safety see the manual entries for...
Safe–Tcl and the `load` Tcl command.

A safe interpreter may not alter the recursion limit of any interpreter, including itself.

**ALIAS INVOCATION**

The alias mechanism has been carefully designed so that it can be used safely when an untrusted script is executing in a safe slave and the target of the alias is a trusted master. The most important thing in guaranteeing safety is to ensure that information passed from the slave to the master is never evaluated or substituted in the master; if this were to occur, it would enable an evil script in the slave to invoke arbitrary functions in the master, which would compromise security.

When the source for an alias is invoked in the slave interpreter, the usual Tcl substitutions are performed when parsing that command. These substitutions are carried out in the source interpreter just as they would be for any other command invoked in that interpreter. The command procedure for the source command takes its arguments and merges them with the `targetCmd` and `args` for the alias to create a new array of arguments. If the words of `srcCmd` were `"srcCmd arg1 arg2 ... argN"`, the new set of words will be `"targetCmd arg arg ... arg arg1 arg2 ... argN"`, where `targetCmd` and `args` are the values supplied when the alias was created. `TargetCmd` is then used to locate a command procedure in the target interpreter, and that command procedure is invoked with the new set of arguments. An error occurs if there is no command named `targetCmd` in the target interpreter. No additional substitutions are performed on the words: the target command procedure is invoked directly, without going through the normal Tcl evaluation mechanism. Substitutions are thus performed on each word exactly once: `targetCmd` and `args` were substituted when parsing the command that created the alias, and `arg1 − argN` are substituted when parsing the command that created the alias, and `arg1 − argN` are substituted when parsing the command that created the alias, and `arg1 − argN` are substituted when parsing the command that created the alias.

When writing the `targetCmds` for aliases in safe interpreters, it is very important that the arguments to that command never be evaluated or substituted, since this would provide an escape mechanism whereby the slave interpreter could execute arbitrary code in the master. This in turn would compromise the security of the system.

**HIDDEN COMMANDS**

Safe interpreters greatly restrict the functionality available to Tcl programs executing within them. Allowing the untrusted Tcl program to have direct access to this functionality is unsafe, because it can be used for a variety of attacks on the environment. However, there are times when there is a legitimate need to use the dangerous functionality in the context of the safe interpreter. For example, sometimes a program must be `source`ed into the interpreter. Another example is Tk, where windows are bound to the hierarchy of windows for a specific interpreter; some potentially dangerous functions, e.g. window management, must be performed on these windows within the interpreter context.

The `interp` command provides a solution to this problem in the form of hidden commands. Instead of removing the dangerous commands entirely from a safe interpreter, these commands are hidden so they become unavailable to Tcl scripts executing in the interpreter. However, such hidden commands can be invoked by any trusted ancestor of the safe interpreter, in the context of the safe interpreter, using `interp`
invoke. Hidden commands and exposed commands reside in separate name spaces. It is possible to define a
hidden command and an exposed command by the same name within one interpreter.

Hidden commands in a slave interpreter can be invoked in the body of procedures called in the master during
alias invocation. For example, an alias for source could be created in a slave interpreter. When it is invoked in
the slave interpreter, a procedure is called in the master interpreter to check that the operation is allowable
(e.g. it asks to source a file that the slave interpreter is allowed to access). The procedure then it invokes the
hidden source command in the slave interpreter to actually source in the contents of the file. Note that two
commands named source exist in the slave interpreter: the alias, and the hidden command.

Because a master interpreter may invoke a hidden command as part of handling an alias invocation, great care
must be taken to avoid evaluating any arguments passed in through the alias invocation. Otherwise, malicious
slave interpreters could cause a trusted master interpreter to execute dangerous commands on their behalf. See
the section on ALIAS INVOCATION for a more complete discussion of this topic. To help avoid this
problem, no substitutions or evaluations are applied to arguments of interp invokehidden.

Safe interpreters are not allowed to invoke hidden commands in themselves or in their descendants. This
prevents safe slaves from gaining access to hidden functionality in themselves or their descendants.

The set of hidden commands in an interpreter can be manipulated by a trusted interpreter using interp expose
and interp hide. The interp expose command moves a hidden command to the set of exposed commands in
the interpreter identified by path, potentially renaming the command in the process. If an exposed command
by the targeted name already exists, the operation fails. Similarly, interp hide moves an exposed command to
the set of hidden commands in that interpreter. Safe interpreters are not allowed to move commands between
the set of hidden and exposed commands, in either themselves or their descendants.

Currently, the names of hidden commands cannot contain namespace qualifiers, and you must first rename a
command in a namespace to the global namespace before you can hide it. Commands to be hidden by interp
hide are looked up in the global namespace even if the current namespace is not the global one. This prevents
slaves from fooling a master interpreter into hiding the wrong command, by making the current namespace be
different from the global one.

CREDITS

This mechanism is based on the Safe−Tcl prototype implemented by Nathaniel Borenstein and Marshall Rose.

EXAMPLES

Creating and using an alias for a command in the current interpreter:

    interp alias {} getIndex {} lsearch {alpha beta gamma delta}
    set idx [getIndex delta]

Executing an arbitrary command in a safe interpreter where every invokation of lappend is logged:

    set i [interp create -safe]
interp hide $i lappend
interp alias $i lappend () loggedLappend $i
proc loggedLappend {i args} {
    puts "logged invocation of lappend $args"
    # Be extremely careful about command construction
    eval [linsert $args 0 \n        interp invokehidden $i lappend]
}
interp eval $i $someUntrustedScript

SEE ALSO

load, safe, Tcl_CreateSlave

KEYWORDS

alias, master interpreter, safe interpreter, slave interpreter

Copyright ' 1995-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
join

NAME

join – Create a string by joining together list elements

SYNOPSIS

join list ?joinString?

DESCRIPTION

The list argument must be a valid Tcl list. This command returns the string formed by joining all of the elements of list together with joinString separating each adjacent pair of elements. The joinString argument defaults to a space character.

EXAMPLES

Making a comma-separated list:

set data {1 2 3 4 5}
join $data ", "
=> 1, 2, 3, 4, 5

Using join to flatten a list by a single level:

set data {1 {2 3} 4 {5 {6 7} 8}}
join $data
=> 1 2 3 4 5 {6 7} 8

SEE ALSO

list, lappend, split

KEYWORDS

element, join, list, separator

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
lappend

NAME

lappend – Append list elements onto a variable

SYNOPSIS

lappend varName ?value value value ...?

DESCRIPTION

This command treats the variable given by varName as a list and appends each of the value arguments to that list as a separate element, with spaces between elements. If varName doesn't exist, it is created as a list with elements given by the value arguments. Lappend is similar to append except that the values are appended as list elements rather than raw text. This command provides a relatively efficient way to build up large lists. For example, "lappend a $b" is much more efficient than "set a [concat $a [list $b]]" when $a is long.

EXAMPLE

Using lappend to build up a list of numbers.

% set var 1
1
% lappend var 2
1 2
% lappend var 3 4 5
1 2 3 4 5

SEE ALSO

list, lindex, linsert, llength, lset(n) Isort, lrange

KEYWORDS

append, element, list, variable
library

NAME

auto_execok, auto_import, auto_load, auto_mkindex, auto_mkindex_old, auto_qualify, auto_reset, tcl_findLibrary, parray, tcl_endOfWord, tcl_startOfNextWord, tcl_startOfPreviousWord, tcl_wordBreakAfter, tcl_wordBreakBefore – standard library of Tcl procedures

SYNOPSIS

auto_execok cmd
auto_import pattern
auto_load cmd
auto_mkindex dir pattern pattern ...
auto_mkindex_old dir pattern pattern ...
auto_qualify command namespace
auto_reset
tcl_findLibrary basename version patch initScript enVarName varName
parray arrayName
tcl_endOfWord str start
tcl_startOfNextWord str start
tcl_startOfPreviousWord str start
tcl_wordBreakAfter str start
tcl_wordBreakBefore str start

INTRODUCTION

COMMAND PROCEDURES

auto_execok cmd
auto_import pattern
auto_load cmd
auto_mkindex dir pattern pattern ...
auto_reset
auto_qualify command namespace
tcl_findLibrary basename version patch initScript enVarName varName
parray arrayName
tcl_endOfWord str start
tcl_startOfNextWord str start
tcl_startOfPreviousWord str start
tcl_wordBreakAfter str start
tcl_wordBreakBefore str start

VARIABLES

auto_execs
auto_index
auto_noexec
auto_noload
auto_path
NAME

auto_execok, auto_import, auto_load, auto_mkindex, auto_mkindex_old, auto_qualify, auto_reset,
tcl_findLibrary, parray, tcl_endOfWord, tcl_startOfNextWord, tcl_startOfPreviousWord,
tcl_wordBreakAfter, tcl_wordBreakBefore – standard library of Tcl procedures

SYNOPSIS

auto_execok cmd
auto_import pattern
auto_load cmd
auto_mkindex dir pattern pattern ...
auto_mkindex_old dir pattern pattern ...
auto_qualify command namespace
auto_reset
tcl_findLibrary basename version patch initScript enVarName varName
parray arrayName
tcl_endOfWord str start
tcl_startOfNextWord str start
tcl_startOfPreviousWord str start
tcl_wordBreakAfter str start
tcl_wordBreakBefore str start

INTRODUCTION

Tcl includes a library of Tcl procedures for commonly-needed functions. The procedures defined in the Tcl library are generic ones suitable for use by many different applications. The location of the Tcl library is returned by the info library command. In addition to the Tcl library, each application will normally have its own library of support procedures as well; the location of this library is normally given by the value of the $app_library global variable, where app is the name of the application. For example, the location of the Tk library is kept in the variable $tk_library.

To access the procedures in the Tcl library, an application should source the file init.tcl in the library, for example with the Tcl command

source [file join [info library] init.tcl]
If the library procedure `Tcl_Init` is invoked from an application's `Tcl_AppInit` procedure, this happens automatically. The code in `init.tcl` will define the `unknown` procedure and arrange for the other procedures to be loaded on-demand using the auto-load mechanism defined below.

**COMMAND PROCEDURES**

The following procedures are provided in the Tcl library:

/auto_execok cmd/
Determines whether there is an executable file or shell builtin by the name `cmd`. If so, it returns a list of arguments to be passed to `exec` to execute the executable file or shell builtin named by `cmd`. If not, it returns an empty string. This command examines the directories in the current search path (given by the PATH environment variable) in its search for an executable file named `cmd`. On Windows platforms, the search is expanded with the same directories and file extensions as used by `exec`. `Auto_exec` remembers information about previous searches in an array named `auto_execs`; this avoids the path search in future calls for the same `cmd`. The command `auto_reset` may be used to force `auto_execok` to forget its cached information.

/auto_import pattern/
`Auto_import` is invoked during namespace import to see if the imported commands specified by `pattern` reside in an autoloaded library. If so, the commands are loaded so that they will be available to the interpreter for creating the import links. If the commands do not reside in an autoloaded library, `auto_import` does nothing. The pattern matching is performed according to the matching rules of namespace import.

/auto_load cmd/
This command attempts to load the definition for a Tcl command named `cmd`. To do this, it searches an auto−load path, which is a list of one or more directories. The auto−load path is given by the global variable `$auto_path` if it exists. If there is no `$auto_path` variable, then the TCLLIBPATH environment variable is used, if it exists. Otherwise the auto−load path consists of just the Tcl library directory. Within each directory in the auto−load path there must be a file `tclIndex` that describes one or more commands defined in that directory and a script to evaluate to load each of the commands. The `tclIndex` file should be generated with the `auto_mkindex` command. If `cmd` is found in an index file, then the appropriate script is evaluated to create the command. The `auto_load` command returns 1 if `cmd` was successfully created. The command returns 0 if there was no index entry for `cmd` or if the script didn't actually define `cmd` (e.g. because index information is out of date). If an error occurs while processing the script, then that error is returned. `Auto_load` only reads the index information once and saves it in the array `auto_index`; future calls to `auto_load` check for `cmd` in the array rather than re−reading the index files. The cached index information may be deleted with the command `auto_reset`. This will force the next `auto_load` command to reload the index database from disk.

/auto_mkindex dir pattern pattern ...
Generates an index suitable for use by `auto_load`. The command searches `dir` for all files whose names match any of the `pattern` arguments (matching is done with the `glob` command), generates an index of all the Tcl command procedures defined in all the matching files, and stores the index information in a file named `tclIndex` in `dir`. If no pattern is given a pattern of `*.tcl` will be assumed. For example, the command

/auto_mkindex foo *.tcl/
will read all the .tcl files in subdirectory foo and generate a new index file foo/tclIndex.

**Auto_mkindex** parses the Tcl scripts by sourcing them into a slave interpreter and monitoring the proc and namespace commands that are executed. Extensions can use the (undocumented) auto_mkindex_parser package to register other commands that can contribute to the auto_load index. You will have to read through auto.tcl to see how this works.

**Auto_mkindex_old** parses the Tcl scripts in a relatively unsophisticated way: if any line contains the word `proc` as its first characters then it is assumed to be a procedure definition and the next word of the line is taken as the procedure's name. Procedure definitions that don't appear in this way (e.g. they have spaces before the `proc`) will not be indexed. If your script contains "dangerous" code, such as global initialization code or procedure names with special characters like $, *, [ or ], you are safer using auto_mkindex_old.

**auto_reset**

Destroys all the information cached by **auto_execok** and **auto_load**. This information will be re-read from disk the next time it is needed. **Auto_reset** also deletes any procedures listed in the auto-load index, so that fresh copies of them will be loaded the next time that they're used.

**auto_qualify** *command* *namespace*

Computes a list of fully qualified names for *command*. This list mirrors the path a standard Tcl interpreter follows for command lookups: first it looks for the command in the current namespace, and then in the global namespace. Accordingly, if *command* is relative and *namespace* is not ::, the list returned has two elements: *command* scoped by *namespace*, as if it were a command in the *namespace* namespace; and *command* as if it were a command in the global namespace. Otherwise, if either *command* is absolute (it begins with ::), or *namespace* is ::, the list contains only *command* as if it were a command in the global namespace.

**Auto_qualify** is used by the auto-loading facilities in Tcl, both for producing auto-loading indexes such as pkgIndex.tcl, and for performing the actual auto-loading of functions at runtime.

**tcl_findLibrary** *basename* *version* *patch* *initScript* *enVarName* *varName*

This is a standard search procedure for use by extensions during their initialization. They call this procedure to look for their script library in several standard directories. The last component of the name of the library directory is normally *basename*version (e.g., tk8.0), but it might be "library" when in the build hierarchies. The *initScript* file will be sourced into the interpreter once it is found. The directory in which this file is found is stored into the global variable *varName*. If this variable is already defined (e.g., by C code during application initialization) then no searching is done. Otherwise the search looks in these directories: the directory named by the environment variable *enVarName*; relative to the Tcl library directory; relative to the executable file in the standard installation bin or bin/arch directory; relative to the executable file in the current build tree; relative to the executable file in a parallel build tree.

**parray** *arrayName*

Prints on standard output the names and values of all the elements in the array *arrayName*.

*ArrayName* must be an array accessible to the caller of **parray**. It may be either local or global.

**tcl_endOfWord** *str* *start*

Returns the index of the first end-of-word location that occurs after a starting index *start* in the string *str*. An end-of-word location is defined to be the first non-word character following the first word.
character after the starting point. Returns −1 if there are no more end−of−word locations after the starting point. See the description of \texttt{tcl\_wordchars} and \texttt{tcl\_nonwordchars} below for more details on how Tcl determines which characters are word characters.

\texttt{tcl\_startOfNextWord} \texttt{str start}
Returns the index of the first start−of−word location that occurs after a starting index \texttt{start} in the string \texttt{str}. A start−of−word location is defined to be the first word character following a non−word character. Returns −1 if there are no more start−of−word locations after the starting point.

\texttt{tcl\_startOfPreviousWord} \texttt{str start}
Returns the index of the first start−of−word location that occurs before a starting index \texttt{start} in the string \texttt{str}. Returns −1 if there are no more start−of−word locations before the starting point.

\texttt{tcl\_wordBreakAfter} \texttt{str start}
Returns the index of the first word boundary after the starting index \texttt{start} in the string \texttt{str}. Returns −1 if there are no more boundaries after the starting point in the given string. The index returned refers to the second character of the pair that comprises a boundary.

\texttt{tcl\_wordBreakBefore} \texttt{str start}
Returns the index of the first word boundary before the starting index \texttt{start} in the string \texttt{str}. Returns −1 if there are no more boundaries before the starting point in the given string. The index returned refers to the second character of the pair that comprises a boundary.

\section*{VARIABLES}

The following global variables are defined or used by the procedures in the Tcl library:

\begin{itemize}
\item \texttt{auto\_execs}
  Used by \texttt{auto\_execok} to record information about whether particular commands exist as executable files.
\item \texttt{auto\_index}
  Used by \texttt{auto\_load} to save the index information read from disk.
\item \texttt{auto\_noexec}
  If set to any value, then \texttt{unknown} will not attempt to auto−exec any commands.
\item \texttt{auto\_noload}
  If set to any value, then \texttt{unknown} will not attempt to auto−load any commands.
\item \texttt{auto\_path}
  If set, then it must contain a valid Tcl list giving directories to search during auto−load operations. This variable is initialized during startup to contain, in order: the directories listed in the TCLLIBPATH environment variable, the directory named by the \texttt{tcl\_library} variable, the parent directory of \texttt{tcl\_library}, the directories listed in the \texttt{tcl\_pkgPath} variable.
\item \texttt{env(TCL\_LIBRARY)}
  If set, then it specifies the location of the directory containing library scripts (the value of this variable will be assigned to the \texttt{tcl\_library} variable and therefore returned by the command \texttt{info\ library}). If this variable isn't set then a default value is used.
\item \texttt{env(TCLLIBPATH)}
  If set, then it must contain a valid Tcl list giving directories to search during auto−load operations. Directories must be specified in Tcl format, using "/" as the path separator, regardless of platform. This variable is only used when initializing the \texttt{auto\_path} variable.
\end{itemize}
**tcl_nonwordchars**

This variable contains a regular expression that is used by routines like `tcl_endOfWord` to identify whether a character is part of a word or not. If the pattern matches a character, the character is considered to be a non-word character. On Windows platforms, spaces, tabs, and newlines are considered non-word characters. Under Unix, everything but numbers, letters and underscores are considered non-word characters.

---

**tcl_wordchars**

This variable contains a regular expression that is used by routines like `tcl_endOfWord` to identify whether a character is part of a word or not. If the pattern matches a character, the character is considered to be a word character. On Windows platforms, words are comprised of any character that is not a space, tab, or newline. Under Unix, words are comprised of numbers, letters or underscores.

---

**unknown_pending**

Used by `unknown` to record the command(s) for which it is searching. It is used to detect errors where `unknown` recurses on itself infinitely. The variable is unset before `unknown` returns.

---

**SEE ALSO**

- `info`
- `re_syntax`

---

**KEYWORDS**

- auto-exec
- auto-load
- library
- unknown
- word
- whitespace

---

*Copyright* © 1991-1993 The Regents of the University of California.

*Copyright* © 1994-1996 Sun Microsystems, Inc.

**lindex**

**NAME**

lindex – Retrieve an element from a list

**SYNOPSIS**

`lindex list ?index...?`

**DESCRIPTION**

The `lindex` command accepts a parameter, `list`, which it treats as a Tcl list. It also accepts zero or more `indices` into the list. The indices may be presented either consecutively on the command line, or grouped in a Tcl list and presented as a single argument.

If no indices are presented, the command takes the form:

```
lindex list
```

or

```
lindex list {}
```

In this case, the return value of `lindex` is simply the value of the `list` parameter.

When presented with a single index, the `lindex` command treats `list` as a Tcl list and returns the `index`'th element from it (0 refers to the first element of the list). In extracting the element, `lindex` observes the same rules concerning braces and quotes and backslashes as the Tcl command interpreter; however, variable substitution and command substitution do not occur. If `index` is negative or greater than or equal to the number of elements in `value`, then an empty string is returned. If `index` has the value `end`, it refers to the last element in the list, and `end−integer` refers to the last element in the list minus the specified integer offset.

If additional `index` arguments are supplied, then each argument is used in turn to select an element from the previous indexing operation, allowing the script to select elements from sublists. The command,

```
lindex $a 1 2 3
```

or

```
lindex $a (1 2 3)
```

is synonymous with

```
lindex [lindex [lindex $a 1] 2] 3
```
EXAMPLES

\begin{verbatim}
lindex {a b c} => a b c
lindex {a b c} {} => a b c
lindex {a b c} 0 => a
lindex {a b c} 2 => c
lindex {a b c} end => c
lindex {a b c} end-1 => b
lindex {{a b c} {d e f} {g h i}} 2 1 => h
lindex {{a b c} {d e f} {g h i}} {2 1} => h
lindex {{{a b} {c d}} {{e f} {g h}}} 1 1 0 => g
lindex {{{a b} {c d}} {{e f} {g h}}} {1 1 0} => g
\end{verbatim}

SEE ALSO

\texttt{list, lappend, linsert, llength, lsearch, lset, lsort, lrange, lreplace}

KEYWORDS

\texttt{element, index, list}

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 2001 by Kevin B. Kenny. All rights reserved.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
**linsert**

**NAME**

linsert – Insert elements into a list

**SYNOPSIS**

linsert list index element ?element element ...?

**DESCRIPTION**

This command produces a new list from `list` by inserting all of the `element` arguments just before the `index`'th element of `list`. Each `element` argument will become a separate element of the new list. If `index` is less than or equal to zero, then the new elements are inserted at the beginning of the list. If `index` has the value `end`, or if it is greater than or equal to the number of elements in the list, then the new elements are appended to the list. `end–integer` refers to the last element in the list minus the specified integer offset.

**EXAMPLE**

Putting some values into a list, first indexing from the start and then indexing from the end, and then chaining them together:

```tcl
set oldList {the fox jumps over the dog}
set midList [linsert $oldList 1 quick]
set newList [linsert $midList end-1 lazy]
# The old lists still exist though...
set newerList [linsert [linsert $oldList end-1 quick] 1 lazy]
```

**SEE ALSO**

list, lappend, lindex, llength, lsearch, lset, lsort, lrange, lreplace

**KEYWORDS**

element, insert, list

---

*Copyright 1993 The Regents of the University of California.*

*Copyright 1994-1996 Sun Microsystems, Inc.*

*Copyright 2001 Kevin B. Kenny. All rights reserved.*

*Copyright 1995-1997 Roger E. Critchlow Jr.*
list

NAME
list – Create a list

SYNOPSIS
list ?arg arg ...?

DESCRIPTION
This command returns a list comprised of all the args, or an empty string if no args are specified. Braces and backslashes get added as necessary, so that the lindex command may be used on the result to re-extract the original arguments, and also so that eval may be used to execute the resulting list, with arg1 comprising the command's name and the other args comprising its arguments. List produces slightly different results than concat: concat removes one level of grouping before forming the list, while list works directly from the original arguments.

EXAMPLE
The command

list a b "c d e " " f {g h}"

will return

a b {c d e } { f {g h}}

while concat with the same arguments will return

a b c d e f {g h}

SEE ALSO
lappend, lindex, linsert, llength, lrange, lreplace, lsearch, lset, lsort

KEYWORDS
element, list

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 2001 Kevin B. Kenny. All rights reserved.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
llength

NAME

llength – Count the number of elements in a list

SYNOPSIS

llength list

DESCRIPTION

Treats list as a list and returns a decimal string giving the number of elements in it.

EXAMPLES

The result is the number of elements:

% llength {a b c d e}
5
% llength {a b c}
3
% llength {}
0

Elements are not guaranteed to be exactly words in a dictionary sense of course, especially when quoting is used:

% llength {a b {c d} e}
4
% llength {a b { } c d e}
6

An empty list is not necessarily an empty string:

% set var { }; puts "[string length $var],[llength $var]"
1,0

SEE ALSO

list, lappend, lindex, linsert, lsearch, lset, lsort, lrange, lreplace

KEYWORDS

element, list, length
NAME

load – Load machine code and initialize new commands.

SYNOPSIS

load fileName
load fileName packageName
load fileName packageName interp

DESCRIPTION

This command loads binary code from a file into the application's address space and calls an initialization procedure in the package to incorporate it into an interpreter. fileName is the name of the file containing the code; its exact form varies from system to system but on most systems it is a shared library, such as a .so file under Solaris or a DLL under Windows. packageName is the name of the package, and is used to compute the name of an initialization procedure. interp is the path name of the interpreter into which to load the package (see the interp manual entry for details); if interp is omitted, it defaults to the interpreter in which the load command was invoked.

Once the file has been loaded into the application's address space, one of two initialization procedures will be invoked in the new code. Typically the initialization procedure will add new commands to a Tcl interpreter. The name of the initialization procedure is determined by packageName and whether or not the target interpreter is a safe one. For normal interpreters the name of the initialization procedure will have the form pkg_Init, where pkg is the same as packageName except that the first letter is converted to upper case and all other letters are converted to lower case. For example, if packageName is foo or FOo, the initialization
procedure's name will be **Foo_Init**.

If the target interpreter is a safe interpreter, then the name of the initialization procedure will be **pkg_SafeInit** instead of **pkg_Init**. The **pkg_SafeInit** function should be written carefully, so that it initializes the safe interpreter only with partial functionality provided by the package that is safe for use by untrusted code. For more information on Safe−Tcl, see the **safe** manual entry.

The initialization procedure must match the following prototype:

```
typedef int Tcl_PackageInitProc(Tcl_Interp *interp);
```

The **interp** argument identifies the interpreter in which the package is to be loaded. The initialization procedure must return **TCL_OK** or **TCL_ERROR** to indicate whether or not it completed successfully; in the event of an error it should set the interpreter's result to point to an error message. The result of the **load** command will be the result returned by the initialization procedure.

The actual loading of a file will only be done once for each **fileName** in an application. If a given **fileName** is loaded into multiple interpreters, then the first **load** will load the code and call the initialization procedure; subsequent **loads** will call the initialization procedure without loading the code again. It is not possible to unload or reload a package.

The **load** command also supports packages that are statically linked with the application, if those packages have been registered by calling the **Tcl_StaticPackage** procedure. If **fileName** is an empty string, then **packageName** must be specified. If **packageName** is omitted or specified as an empty string, Tcl tries to guess the name of the package. This may be done differently on different platforms. The default guess, which is used on most UNIX platforms, is to take the last element of **fileName**, strip off the first three characters if they are **lib**, and use any following alphabetic and underline characters as the module name. For example, the command **load libxyz4.2.so** uses the module name **xyz** and the command **load bin/last.so {}** uses the module name **last**.

If **fileName** is an empty string, then **packageName** must be specified. The **load** command first searches for a statically loaded package (one that has been registered by calling the **Tcl_StaticPackage** procedure) by that name; if one is found, it is used. Otherwise, the **load** command searches for a dynamically loaded package by that name, and uses it if it is found. If several different files have been **loaded** with different versions of the package, Tcl picks the file that was loaded first.

**PORTABILITY ISSUES**

**Windows**

When a load fails with "library not found" error, it is also possible that a dependent library was not found. To see the dependent libraries, type ```dumpbin –imports <dllname>``" in a DOS console to see what the library must import. When loading a DLL in the current directory, Windows will ignore ```/``" as a path specifier and use a search heuristic to find the DLL instead. To avoid this, load the DLL with:
load [file join [pwd] mylib.DLL]

BUGS

If the same file is loaded by different fileNames, it will be loaded into the process's address space multiple times. The behavior of this varies from system to system (some systems may detect the redundant loads, others may not).

EXAMPLE

The following is a minimal extension:

```c
#include <tcl.h>
#include <stdio.h>
static int fooCmd(ClientData clientData,
    Tcl_Interp *interp, int objc, char * CONST objv[]) {
    printf("called with %d arguments\n", objc);
    return TCL_OK;
}
int Foo_Init(Tcl_Interp *interp) {
    if (Tcl_InitStubs(interp, "8.1", 0) == NULL) {
        return TCL_ERROR;
    }
    printf("creating foo command");
    Tcl_CreateObjCommand(interp, "foo", fooCmd, NULL, NULL);
    return TCL_OK;
}
```

When built into a shared/dynamic library with a suitable name (e.g. foo.dll on Windows, libfoo.so on Solaris and Linux) it can then be loaded into Tcl with the following:

```bash
# Load the extension
switch $tcl_platform(platform) {
    windows {
        load ./foo.dll
    }
    unix {
        load ./libfoo[info sharedlibextension]
    }
}
# Now execute the command defined by the extension
foo
```

SEE ALSO

info sharedlibextension, Tcl_StaticPackage, safe
KEYWORDS

binary code, loading, safe interpreter, shared library

Copyright © 1995−1996 Sun Microsystems, Inc.
Copyright © 1995−1997 Roger E. Critchlow Jr.
lrange

NAME

lrange – Return one or more adjacent elements from a list

SYNOPSIS

lrange list first last

DESCRIPTION

List must be a valid Tcl list. This command will return a new list consisting of elements first through last, inclusive. First or last may be end (or any abbreviation of it) to refer to the last element of the list. If first is less than zero, it is treated as if it were zero. If last is greater than or equal to the number of elements in the list, then it is treated as if it were end. If first is greater than last then an empty string is returned. Note: "lrange list first first" does not always produce the same result as "lindex list first" (although it often does for simple fields that aren't enclosed in braces); it does, however, produce exactly the same results as "list [lindex list first]"

EXAMPLES

Selecting the first two elements:

% lrange (a b c d e) 0 1
a b

Selecting the last three elements:

% lrange (a b c d e) end−2 end
c d e

Selecting everything except the first and last element:

% lrange (a b c d e) 1 end−1
b c d

Selecting a single element with lrange is not the same as doing so with lindex:

% set var {some {elements to} select}
some {elements to} select
% lindex $var 1
elements to
% lrange $var 1 1
(elements to)
SEE ALSO

list, lappend, lindex, linsert, llengt, lsearch, lset, lreplace, lsort

KEYWORDS

element, list, range, sublist
Ireplace

NAME

Ireplace – Replace elements in a list with new elements

SYNOPSIS

Ireplace list first last ?element element ...?

DESCRIPTION

Ireplace returns a new list formed by replacing one or more elements of list with the element arguments. first and last specify the first and last index of the range of elements to replace. 0 refers to the first element of the list, and end (or any abbreviation of it) may be used to refer to the last element of the list. If list is empty, then first and last are ignored. If first is less than zero, it is considered to refer to the first element of the list. For non-empty lists, the element indicated by first must exist. If last is less than zero but greater than first, then any specified elements will be prepended to the list. If last is less than first then no elements are deleted; the new elements are simply inserted before first. The element arguments specify zero or more new arguments to be added to the list in place of those that were deleted. Each element argument will become a separate element of the list. If no element arguments are specified, then the elements between first and last are simply deleted. If list is empty, any element arguments are added to the end of the list.

EXAMPLES

Replacing an element of a list with another:

% Ireplace {a b c d e} 1 1 foo
a foo c d e

Replacing two elements of a list with three:

% Ireplace {a b c d e} 1 2 three more elements
a three more elements d e

Deleting the last element from a list in a variable:

% set var {a b c d e}
% set var [Ireplace $var end end]
a b c d

Ireplace
SEE ALSO

list, lappend, lindex, linsert, llen, lsearch, lset, lrange, lsort

KEYWORDS

element, list, replace
NAME

lsearch – See if a list contains a particular element

SYNOPSIS

lsearch ?options? list pattern

DESCRIPTION

This command searches the elements of list to see if one of them matches pattern. If so, the command returns the index of the first matching element (unless the options –all or –inline are specified.) If not, the command returns −1. The option arguments indicates how the elements of the list are to be matched against pattern and it must have one of the following values:

–all

EXAMPLES

SEE ALSO

KEYWORDS
Changes the result to be the list of all matching indices (or all matching values if –inline is specified as well.)

−ascii
The list elements are to be examined as Unicode strings (the name is for backward–compatibility reasons.) This option is only meaningful when used with –exact or –sorted.

−decreasing
The list elements are sorted in decreasing order. This option is only meaningful when used with –sorted.

−dictionary
The list elements are to be compared using dictionary–style comparisons (see lsort for a fuller description). This option is only meaningful when used with –exact or –sorted, and it is only distinguishable from the –ascii option when the –sorted option is given, because values are only dictionary–equal when exactly equal.

−exact
The list element must contain exactly the same string as pattern.

−glob
Pattern is a glob–style pattern which is matched against each list element using the same rules as the string match command.

−increasing
The list elements are sorted in increasing order. This option is only meaningful when used with –sorted.

−inline
The matching value is returned instead of its index (or an empty string if no value matches.) If –all is also specified, then the result of the command is the list of all values that matched.

−integer
The list elements are to be compared as integers. This option is only meaningful when used with –exact or –sorted.

−not
This negates the sense of the match, returning the index of the first non–matching value in the list.

−real
The list elements are to be compared as floating–point values. This option is only meaningful when used with –exact or –sorted.

−regexp
Pattern is treated as a regular expression and matched against each list element using the rules described in the re_syntax reference page.

−sorted
The list elements are in sorted order. If this option is specified, lsearch will use a more efficient searching algorithm to search list. If no other options are specified, list is assumed to be sorted in increasing order, and to contain ASCII strings. This option is mutually exclusive with –glob and –regexp, and is treated exactly like –exact when either –all, or –not is specified.

−start index
The list is searched starting at position index. If index has the value end, it refers to the last element in the list, and end–integer refers to the last element in the list minus the specified integer offset.

If option is omitted then it defaults to –glob. If more than one of –exact, –glob, –regexp, and –sorted is
specified, whichever option is specified last takes precedence. If more than one of \texttt{−ascii}, \texttt{−dictionary}, \texttt{−integer} and \texttt{−real} is specified, the option specified last takes precedence. If more than one of \texttt{−increasing} and \texttt{−decreasing} is specified, the option specified last takes precedence.

\textbf{EXAMPLES}

\begin{verbatim}
lsearch {a b c d e} c => 2
lsearch -all {a b c a b c} c => 2 5
lsearch -inline (a20 b35 c47) b* => b35
lsearch -inline -not (a20 b35 c47) b* => a20
lsearch -all -inline -not (a20 b35 c47) b* => a20 c47
lsearch -all -not (a20 b35 c47) b* => 0 2
lsearch -start 3 {a b c a b c} c => 5
\end{verbatim}

\textbf{SEE ALSO}

\texttt{foreach, list, lappend, lindex, linsert, llength, lset, lsort, lrange, lreplace}

\textbf{KEYWORDS}

\texttt{list, match, pattern, regular expression, search, string}

\textcopyright{} 1993 The Regents of the University of California.
\textcopyright{} 1994-1996 Sun Microsystems, Inc.
\textcopyright{} 2001 Kevin B. Kenny. All rights reserved.
\textcopyright{} 1995-1997 Roger E. Critchlow Jr.
lset

NAME

lset – Change an element in a list

SYNOPSIS

lset varName ?index...? newValue

DESCRIPTION

The lset command accepts a parameter, varName, which it interprets as the name of a variable containing a Tcl list. It also accepts zero or more indices into the list. The indices may be presented either consecutively on the command line, or grouped in a Tcl list and presented as a single argument. Finally, it accepts a new value for an element of varName.

If no indices are presented, the command takes the form:

lset varName newValue

or

lset varName {} newValue

In this case, newValue replaces the old value of the variable varName.

When presented with a single index, the lset command treats the content of the varName variable as a Tcl list. It addresses the index'th element in it (0 refers to the first element of the list). When interpreting the list, lset observes the same rules concerning braces and quotes and backslashes as the Tcl command interpreter; however, variable substitution and command substitution do not occur. The command constructs a new list in which the designated element is replaced with newValue. This new list is stored in the variable varName, and is also the return value from the lset command.

If index is negative or greater than or equal to the number of elements in $varName, then an error occurs.

If index has the value end, it refers to the last element in the list, and end–integer refers to the last element in the list minus the specified integer offset.

If additional index arguments are supplied, then each argument is used in turn to address an element within a sublist designated by the previous indexing operation, allowing the script to alter elements in sublists. The command,

lset a 1 2 newValue
or

\[ \text{lset a \{1 2\} newValue} \]

replaces element 2 of sublist 1 with \textit{newValue}.

The integer appearing in each \textit{index} argument must be greater than or equal to zero. The integer appearing in each \textit{index} argument must be strictly less than the length of the corresponding list. In other words, the \texttt{lset} command cannot change the size of a list. If an index is outside the permitted range, an error is reported.

\section*{EXAMPLES}

In each of these examples, the initial value of \( x \) is:

\begin{verbatim}
set x [list [list a b c] [list d e f] [list g h i]]
=> {a b c} {d e f} {g h i}
\end{verbatim}

The indicated return value also becomes the new value of \( x \) (except in the last case, which is an error which leaves the value of \( x \) unchanged.)

\begin{verbatim}
lset x \{j k l\} => j k l
lset x \{\} \{j k l\} => j k l
lset x 0 j => j \{d e f\} \{g h i\}
lset x 2 j => \{a b c\} \{d e f\} j
lset x end j => \{a b c\} \{d e f\} j
lset x end-1 j => \{a b c\} j \{g h i\}
lset x 2 1 j => \{a b c\} \{d e f\} \{g j i\}
lset x \{2 1\} j => \{a b c\} \{d e f\} \{g j i\}
lset x \{2 3\} j => list index out of range
\end{verbatim}

In the following examples, the initial value of \( x \) is:

\begin{verbatim}
set x [list [list [list a b] [list c d]] \[list [list e f] [list g h]]]
=> {{a b} {c d}} {{e f} {g h}}
\end{verbatim}

The indicated return value also becomes the new value of \( x \).

\begin{verbatim}
lset x \{1 1 0\} j => {{a b} {c d}} {{e f} {j h}}
lset x \{1 1 0\} j => {{a b} {c d}} {{e f} {j h}}
\end{verbatim}

\section*{SEE ALSO}
\texttt{list, lappend, lindex, linsert, llength, lsearch, lsort, lrange, lreplace}
KEYWORDS

element, index, list, replace, set

Copyright ' 2001 by Kevin B. Kenny. All rights reserved.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME
lsort – Sort the elements of a list

SYNOPSIS
lsort ?options? list

DESCRIPTION
This command sorts the elements of list, returning a new list in sorted order. The implementation of the lsort command uses the merge–sort algorithm which is a stable sort that has O(n log n) performance characteristics.

By default ASCII sorting is used with the result returned in increasing order. However, any of the following options may be specified before list to control the sorting process (unique abbreviations are accepted):

−ascii
Use string comparison with Unicode code–point collation order (the name is for backward–compatibility reasons.) This is the default.

−dictionary

SEE ALSO

NOTES

EXAMPLES

KEYWORDS
Use dictionary-style comparison. This is the same as \textit{--ascii} except (a) case is ignored except as a tie-breaker and (b) if two strings contain embedded numbers, the numbers compare as integers, not characters. For example, in \textit{--dictionary} mode, \texttt{bigBoy} sorts between \texttt{bigbang} and \texttt{bigboy}, and \texttt{x10y} sorts between \texttt{x9y} and \texttt{x11y}.

\textit{--integer}

Convert list elements to integers and use integer comparison.

\textit{--real}

Convert list elements to floating-point values and use floating comparison.

\textit{--command \texttt{command}}

Use \texttt{command} as a comparison command. To compare two elements, evaluate a Tcl script consisting of \texttt{command} with the two elements appended as additional arguments. The script should return an integer less than, equal to, or greater than zero if the first element is to be considered less than, equal to, or greater than the second, respectively.

\textit{--increasing}

Sort the list in increasing order ("smallest" items first). This is the default.

\textit{--decreasing}

Sort the list in decreasing order ("largest" items first).

\textit{--index \texttt{index}}

If this option is specified, each of the elements of \texttt{list} must itself be a proper Tcl sublist. Instead of sorting based on whole sublists, \texttt{lsort} will extract the \texttt{index}'th element from each sublist and sort based on the given element. The keyword \texttt{end} is allowed for the \texttt{index} to sort on the last sublist element, and \texttt{end--index} sorts on a sublist element offset from the end. For example,

\begin{verbatim}
lsort --integer --index 1 {{First 24} {Second 18} {Third 30}}
\end{verbatim}

returns \texttt{(Second 18) (First 24) (Third 30)}, and

\begin{verbatim}
lsort --index end-1 {{a 1 e i} {b 2 3 f g} {c 4 5 6 d h}}
\end{verbatim}

returns \texttt{c 4 5 6 d h} \texttt{a 1 e i} \texttt{b 2 3 f g}. This option is much more efficient than using \textit{--command} to achieve the same effect.

\textit{--unique}

If this option is specified, then only the last set of duplicate elements found in the list will be retained. Note that duplicates are determined relative to the comparison used in the sort. Thus if \textit{--index 0} is used, \texttt{\{1 a\}} and \texttt{\{1 b\}} would be considered duplicates and only the second element, \texttt{\{1 b\}}, would be retained.

\textbf{NOTES}

The options to \texttt{lsort} only control what sort of comparison is used, and do not necessarily constrain what the values themselves actually are. This distinction is only noticeable when the list to be sorted has fewer than two elements.

The \texttt{lsort} command is reentrant, meaning it is safe to use as part of the implementation of a command used in the \textit{--command} option.
EXAMPLES

Sorting a list using ASCII sorting:

% lsort {a10 B2 b1 a1 a2}
   B2 a1 a10 a2 b1

Sorting a list using Dictionary sorting:

% lsort -dictionary {a10 B2 b1 a1 a2}
   a1 a2 a10 b1 B2

Sorting lists of integers:

% lsort -integer {5 3 1 2 11 4}
   1 2 3 4 5 11
% lsort -integer {1 2 0x5 7 0 4 -1}
   -1 0 1 2 4 0x5 7

Sorting lists of floating-point numbers:

% lsort -real {5 3 1 2 11 4}
   1 2 3 4 5 11
% lsort -real {.5 0.07e1 0.4 6e−1}
   0.4 .5 6e−1 0.07e1

Sorting using indices:

% # Note the space character before the c
% lsort {{a 5} {c 3} {b 4} {e 1} {d 2}}
{c 3} {a 5} {b 4} {d 2} {e 1}
% lsort -index 0 {{a 5} {c 3} {b 4} {e 1} {d 2}}
{a 5} {b 4} {c 3} {d 2} {e 1}
% lsort -index 1 {{a 5} {c 3} {b 4} {e 1} {d 2}}
{e 1} {d 2} {c 3} {b 4} {a 5}

Stripping duplicate values using sorting:

% lsort -unique {a b c a b c a b c}
   a b c

More complex sorting using a comparison function:

% proc compare {a b} {
   set a0 [lindex $a 0]
   set b0 [lindex $b 0]
   if {$a0 < $b0} {
      return -1
   } elseif {$a0 > $b0} {
      return 1
   } else { return 0 }
return [string compare [lindex $a 1] [lindex $b 1]]
%
lsort -command compare 
   {{3 apple} {0x2 carrot} {1 dingo} {2 banana}}
   {1 dingo} {2 banana} {0x2 carrot} {3 apple}

SEE ALSO

list, lappend, lindex, linsert, llength, lsearch, lset, lrange, lreplace

KEYWORDS

element, list, order, sort

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1999 Scripits Corporation
Copyright ' 2001 Kevin B. Kenny. All rights reserved.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
memory

NAME

memory – Control Tcl memory debugging capabilities.

SYNOPSIS

memory option ?arg arg ...?

DESCRIPTION

memory active file
memory break_on_malloc count
memory info
memory init [on|off]
memory onexit file
memory tag string
memory trace [on|off]
memory trace_on_at_malloc count
memory validate [on|off]

SEE ALSO

KEYWORDS

NAME

memory – Control Tcl memory debugging capabilities.

SYNOPSIS

memory option ?arg arg ...?

DESCRIPTION

The memory command gives the Tcl developer control of Tcl's memory debugging capabilities. The memory command has several suboptions, which are described below. It is only available when Tcl has been compiled with memory debugging enabled (when TCL_MEM_DEBUG is defined at compile time), and after Tcl_InitMemory has been called.

memory active file
Write a list of all currently allocated memory to the specified file.

memory break_on_malloc count
After the count allocations have been performed, ckalloc outputs a message to this effect and that it is now attempting to enter the C debugger. Tcl will then issue a SIGINT signal against itself. If you are running Tcl under a C debugger, it should then enter the debugger command mode.

memory info
Returns a report containing the total allocations and frees since Tcl began, the current packets allocated (the current number of calls to \texttt{ckalloc} not met by a corresponding call to \texttt{ckfree}), the current bytes allocated, and the maximum number of packets and bytes allocated.

\textbf{memory init [on|off]}

Turn on or off the pre-initialization of all allocated memory with bogus bytes. Useful for detecting the use of uninitialized values.

\textbf{memory onexit file}

Causes a list of all allocated memory to be written to the specified file during the finalization of Tcl's memory subsystem. Useful for checking that memory is properly cleaned up during process exit.

\textbf{memory tag string}

Each packet of memory allocated by \texttt{ckalloc} can have associated with it a string–valued tag. In the lists of allocated memory generated by \texttt{memory active} and \texttt{memory onexit}, the tag for each packet is printed along with other information about the packet. The \texttt{memory tag} command sets the tag value for subsequent calls to \texttt{ckalloc} to be \texttt{string}.

\textbf{memory trace [on|off]}

Turns memory tracing on or off. When memory tracing is on, every call to \texttt{ckalloc} causes a line of trace information to be written to \texttt{stderr}, consisting of the word \texttt{ckalloc}, followed by the address returned, the amount of memory allocated, and the C filename and line number of the code performing the allocation. For example:

\begin{verbatim}
ckalloc 40e478 98 tclProc.c 1406
\end{verbatim}

Calls to \texttt{ckfree} are traced in the same manner.

\textbf{memory trace_on_at_malloc count}

Enable memory tracing after \texttt{count} \texttt{ckalloc}'s have been performed. For example, if you enter \texttt{memory trace_on_at_malloc 100}, after the 100th call to \texttt{ckalloc}, memory trace information will begin being displayed for all allocations and frees. Since there can be a lot of memory activity before a problem occurs, judicious use of this option can reduce the slowdown caused by tracing (and the amount of trace information produced), if you can identify a number of allocations that occur before the problem sets in. The current number of memory allocations that have occurred since Tcl started is printed on a guard zone failure.

\textbf{memory validate [on|off]}

Turns memory validation on or off. When memory validation is enabled, on every call to \texttt{ckalloc} or \texttt{ckfree}, the guard zones are checked for every piece of memory currently in existence that was allocated by \texttt{ckalloc}. This has a large performance impact and should only be used when overwrite problems are strongly suspected. The advantage of enabling memory validation is that a guard zone overwrite can be detected on the first call to \texttt{ckalloc} or \texttt{ckfree} after the overwrite occurred, rather than when the specific memory with the overwritten guard zone(s) is freed, which may occur long after the overwrite occurred.

\textbf{SEE ALSO}

\texttt{ckalloc}, \texttt{ckfree}, \texttt{Tcl\_ValidateAllMemory}, \texttt{Tcl\_DumpActiveMemory}, \texttt{TCL\_MEM\_DEBUG}
msgcat

NAME

msgcat – Tcl message catalog

SYNOPSIS

package require Tcl 8.2
package require msgcat 1.3
::msgcat::mc src-string ?arg arg ...?
::msgcat::mcmmax ?src–string src–string ...?
::msgcat::mlocale ?newLocale?
::msgcat::mcpreferences
::msgcat::mcload dirname
::msgcat::mcset locale src–string ?translate–string?
::msgcat::mcmset locale src–trans–list
::msgcat::mcunknown locale src–string

DESCRIPTION

COMMANDS

::msgcat::mc src–string ?arg arg ...?
::msgcat::mcmmax ?src–string src–string ...?
::msgcat::mlocale ?newLocale?
::msgcat::mcpreferences
::msgcat::mcload dirname
::msgcat::mcset locale src–string ?translate–string?
::msgcat::mcmset locale src–trans–list
::msgcat::mcunknown locale src–string

LOCALE SPECIFICATION

NAMESPACE SPECIFICATION

LOCATION AND FORMAT OF MESSAGE FILES

RECOMMENDED MESSAGE SETUP FOR PACKAGES

POSITIONAL CODES FOR FORMAT AND SCAN COMMANDS

CREDITS

SEE ALSO

KEYWORDS
::msgcat::mc src−string ?arg arg ...?
::msgcat::mcmmax ?src−string src−string ...
::msgcat::mclocale ?newLocale?
::msgcat::mcpreferences
::msgcat::mcload dirname
::msgcat::mcset locale src−string ?translate−string?
::msgcat::mcset locale src−trans−list
::msgcat::mcunknown locale src−string

DESCRIPTION

The msgcat package provides a set of functions that can be used to manage multi-lingual user interfaces. Text strings are defined in a "message catalog" which is independent from the application, and which can be edited or localized without modifying the application source code. New languages or locales are provided by adding a new file to the message catalog.

Use of the message catalog is optional by any application or package, but is encouraged if the application or package wishes to be enabled for multi-lingual applications.

COMMANDS

::msgcat::mc src−string ?arg arg ...?

Returns a translation of src−string according to the user's current locale. If additional arguments past src−string are given, the format command is used to substitute the additional arguments in the translation of src−string.

::msgcat::mc will search the messages defined in the current namespace for a translation of src−string; if none is found, it will search in the parent of the current namespace, and so on until it reaches the global namespace. If no translation string exists, ::msgcat::mcunknown is called and the string returned from ::msgcat::mcunknown is returned.

::msgcat::mc is the main function used to localize an application. Instead of using an English string directly, an application can pass the English string through ::msgcat::mc and use the result. If an application is written for a single language in this fashion, then it is easy to add support for additional languages later simply by defining new message catalog entries.

::msgcat::mcmmax ?src−string src−string ...

Given several source strings, ::msgcat::mcmmax returns the length of the longest translated string. This is useful when designing localized GUIs, which may require that all buttons, for example, be a fixed width (which will be the width of the widest button).

::msgcat::mclocale ?newLocale?

This function sets the locale to newLocale. If newLocale is omitted, the current locale is returned, otherwise the current locale is set to newLocale. msgcat stores and compares the locale in a case-insensitive manner, and returns locales in lowercase. The initial locale is determined by the locale specified in the user's environment. See LOCALE SPECIFICATION below for a description
of the locale string format.

::msgcat::mcpreferences
Returns an ordered list of the locales preferred by the user, based on the user's language specification. The list is ordered from most specific to least preference. The list is derived from the current locale set in msgcat by ::msgcat::mclocale, and cannot be set independently. For example, if the current locale is en_US_funky, then ::msgcat::mcpreferences returns {en_US_funky en_US en}.

::msgcat::mcload dirname
Searches the specified directory for files that match the language specifications returned by ::msgcat::mcpreferences (note that these are all lowercase), extended by the file extension `.msg". Each matching file is read in order, assuming a UTF-8 encoding. The file contents are then evaluated as a Tcl script. This means that Unicode characters may be present in the message file either directly in their UTF-8 encoded form, or by use of the backslash-u quoting recognized by Tcl evaluation. The number of message files which matched the specification and were loaded is returned.

::msgcat::mcset locale src-string ?translate-string?
Sets the translation for src-string to translate-string in the specified locale and the current namespace. If translate-string is not specified, src-string is used for both. The function returns translate-string.

::msgcat::mcset locale src-trans-list
Sets the translation for multiple source strings in src-trans-list in the specified locale and the current namespace. src-trans-list must have an even number of elements and is in the form {src-string translate-string ?src-string translate-string ...?} ::msgcat::mcset can be significantly faster than multiple invocations of ::msgcat::mcset. The function returns the number of translations set.

::msgcat::mcunknown locale src-string
This routine is called by ::msgcat::mc in the case when a translation for src-string is not defined in the current locale. The default action is to return src-string. This procedure can be redefined by the application, for example to log error messages for each unknown string. The ::msgcat::mcunknown procedure is invoked at the same stack context as the call to ::msgcat::mc. The return value of ::msgcat::mcunknown is used as the return value for the call to ::msgcat::mc.

LOCALE SPECIFICATION

The locale is specified to msgcat by a locale string passed to ::msgcat::mclocale. The locale string consists of a language code, an optional country code, and an optional system-specific code, each separated by `_". The country and language codes are specified in standards ISO-639 and ISO-3166. For example, the locale `en" specifies English and `en_US" specifies U.S. English.

When the msgcat package is first loaded, the locale is initialized according to the user's environment. The variables env(LC_ALL), env(LC_MESSAGES), and env(LANG) are examined in order. The first of them to have a non-empty value is used to determine the initial locale. The value is parsed according to the XPG4 pattern

language[_country][.codeset]@[modifier]

to extract its parts. The initial locale is then set by calling ::msgcat::mclocale with the argument

language[_country] [.modifier]
On Windows, if none of those environment variables is set, msgcat will attempt to extract locale information from the registry. If all these attempts to discover an initial locale from the user's environment fail, msgcat defaults to an initial locale of ``C''.

When a locale is specified by the user, a ``best match'' search is performed during string translation. For example, if a user specifies en_GB_Funky, the locales ``en_GB_Funky'', ``en_GB'', and ``en'' are searched in order until a matching translation string is found. If no translation string is available, then ::msgcat::unknown is called.

**NAMESPACE AND MESSAGE CATALOGS**

Strings stored in the message catalog are stored relative to the namespace from which they were added. This allows multiple packages to use the same strings without fear of collisions with other packages. It also allows the source string to be shorter and less prone to typographical error.

For example, executing the code

```tcl
::msgcat::mcset en hello "hello from ::"
namespace eval foo {
    ::msgcat::mcset en hello "hello from ::foo"
}
puts ::msgcat::mc hello
namespace eval foo {puts ::msgcat::mc hello}
```

will print

```
hello from ::
hello from ::foo
```

When searching for a translation of a message, the message catalog will search first the current namespace, then the parent of the current namespace, and so on until the global namespace is reached. This allows child namespaces to "inherit" messages from their parent namespace.

For example, executing (in the ``en'' locale) the code

```tcl
::msgcat::mcset en m1 ":: message1"
::msgcat::mcset en m2 ":: message2"
::msgcat::mcset en m3 ":: message3"
namespace eval ::foo {
    ::msgcat::mcset en m2 "::foo message2"
    ::msgcat::mcset en m3 "::foo message3"
}
namespace eval ::foo::bar {
    ::msgcat::mcset en m3 "::foo::bar message3"
}
namespace import ::msgcat::mc
puts "[mc m1]; [mc m2]; [mc m3]"
namespace eval ::foo {puts "[mc m1]; [mc m2]; [mc m3]"}
namespace eval ::foo::bar {puts "[mc m1]; [mc m2]; [mc m3]"}
```

**NAMESPACE AND MESSAGE CATALOGS**

---

Tcl8.4/Tk8.4 Manual

---
will print

:: message1; :: message2; :: message3
:: message1; ::foo message2; ::foo message3
:: message1; ::foo message2; ::foo::bar message3

LOCATION AND FORMAT OF MESSAGE FILES

Message files can be located in any directory, subject to the following conditions:

[1] All message files for a package are in the same directory.
[2] The message file name is a msgcat locale specifier (all lowercase) followed by `.msg`. For example:

```
es.msg    -- spanish
en_gb.msg -- United Kingdom English
```

[3] The file contains a series of calls to `mcset` and `mcmset`, setting the necessary translation strings for the language, likely enclosed in a `namespace eval` so that all source strings are tied to the namespace of the package. For example, a short `es.msg` might contain:

```
namespace eval ::mypackage {
    ::msgcat::mcset es "Free Beer!" "Cerveza Gracias!"
}
```

RECOMMENDED MESSAGE SETUP FOR PACKAGES

If a package is installed into a subdirectory of the `tcl_pkgPath` and loaded via `package require`, the following procedure is recommended.

[1] During package installation, create a subdirectory `msgs` under your package directory.
[3] Add the following command to your package initialization script:

```
# load language files, stored in msgs subdirectory
::msgcat::mcload [file join [file dirname [info script]]] msgs
```

POSITIONAL CODES FOR FORMAT AND SCAN COMMANDS

It is possible that a message string used as an argument to `format` might have positionally dependent parameters that might need to be repositioned. For example, it might be syntactically desirable to rearrange the sentence structure while translating.
format "We produced %d units in location %s" $num $city
format "In location %s we produced %d units" $city $num

This can be handled by using the positional parameters:

format "We produced %1\$d units in location %2\$s" $num $city
format "In location %2\$s we produced %1\$d units" $num $city

Similarly, positional parameters can be used with scan to extract values from internationalized strings.

CREDITS

The message catalog code was developed by Mark Harrison.

SEE ALSO

format, scan, namespace, package

KEYWORDS

internationalization, i18n, localization, l10n, message, text, translation
namespace

NAME

namespace – create and manipulate contexts for commands and variables

SYNOPSIS

namespace ?option? ?arg ...?

DESCRIPTION

namespace children ?namespace? ?pattern?
namespace code script
namespace current
namespace delete ?namespace namespace ...?
namespace eval namespace arg ?arg ...?
namespace exists namespace
namespace export ?--clear? ?pattern pattern ...
namespace forget ?pattern pattern ...
namespace import ?--force? ?pattern pattern ...
namespace inscope namespace script ?arg ...
namespace origin command
namespace parent ?namespace?
namespace qualifiers string
namespace tail string
namespace which ?--command? ?--variable? name

WHAT IS A NAMESPACE?
QUALIFIED NAMES
NAME RESOLUTION
IMPORTING COMMANDS
EXPORTING COMMANDS
SCOPED SCRIPTS
EXAMPLES
SEE ALSO
KEYWORDS
DESCRIPTION

The `namespace` command lets you create, access, and destroy separate contexts for commands and variables. See the section **WHAT IS A NAMESPACE?** below for a brief overview of namespaces. The legal values of `option` are listed below. Note that you can abbreviate the `options`.

**namespace children ?namespace? ?pattern?**

Returns a list of all child namespaces that belong to the namespace `namespace`. If `namespace` is not specified, then the children are returned for the current namespace. This command returns fully-qualified names, which start with a double colon (`::`). If the optional `pattern` is given, then this command returns only the names that match the glob-style pattern. The actual pattern used is determined as follows: a pattern that starts with double colon (`::`) is used directly, otherwise the namespace `namespace` (or the fully-qualified name of the current namespace) is prepended onto the pattern.

**namespace code script**

Captures the current namespace context for later execution of the script `script`. It returns a new script in which `script` has been wrapped in a `namespace inscope` command. The new script has two important properties. First, it can be evaluated in any namespace and will cause `script` to be evaluated in the current namespace (the one where the `namespace code` command was invoked). Second, additional arguments can be appended to the resulting script and they will be passed to `script` as additional arguments. For example, suppose the command `set script [namespace code {foo bar}]` is invoked in namespace `::a::b`. Then `eval "$script x y"` can be executed in any namespace (assuming the value of `script` has been passed in properly) and will have the same effect as the command `::namespace eval ::a::b [foo bar x y]`. This command is needed because extensions like Tk normally execute callback scripts in the global namespace. A scoped command captures a command together with its namespace context in a way that allows it to be executed properly later. See the section **SCOPED SCRIPTS** for some examples of how this is used to create callback scripts.

**namespace current**

Returns the fully-qualified name for the current namespace. The actual name of the global namespace is `` (i.e., an empty string), but this command returns `::` for the global namespace as a convenience to programmers.

**namespace delete ?namespace namespace ...?**

Each namespace `namespace` is deleted and all variables, procedures, and child namespaces contained in the namespace are deleted. If a procedure is currently executing inside the namespace, the namespace will be kept alive until the procedure returns; however, the namespace is marked to prevent other code from looking it up by name. If a namespace doesn't exist, this command returns an error. If no namespace names are given, this command does nothing.

**namespace eval namespace arg ?arg ...?**

Activates a namespace called `namespace` and evaluates some code in that context. If the namespace does not already exist, it is created. If more than one `arg` argument is specified, the arguments are concatenated together with a space between each one in the same fashion as the `eval` command, and the result is evaluated.

If `namespace` has leading namespace qualifiers and any leading namespaces do not exist, they are automatically created.
namespace exists namespace

    Returns 1 if namespace is a valid namespace in the current context, returns 0 otherwise.

namespace export ?–clear? ?pattern pattern ...?

    Specifies which commands are exported from a namespace. The exported commands are those that can be later imported into another namespace using a namespace import command. Both commands defined in a namespace and commands the namespace has previously imported can be exported by a namespace. The commands do not have to be defined at the time the namespace export command is executed. Each pattern may contain glob−style special characters, but it may not include any namespace qualifiers. That is, the pattern can only specify commands in the current (exporting) namespace. Each pattern is appended onto the namespace's list of export patterns. If the –clear flag is given, the namespace's export pattern list is reset to empty before any pattern arguments are appended. If no patterns are given and the –clear flag isn't given, this command returns the namespace's current export list.

namespace forget ?pattern pattern ...?

    Removes previously imported commands from a namespace. Each pattern is a simple or qualified name such as x, foo::x or a::b::p*. Qualified names contain double colons (::) and qualify a name with the name of one or more namespaces. Each qualified pattern is qualified with the name of an exporting namespace and may have glob−style special characters in the command name at the end of the qualified name. Glob characters may not appear in a namespace name. For each simple pattern this command deletes the matching commands of the current namespace that were imported from a different namespace. For qualified patterns, this command first finds the matching exported commands. It then checks whether any of those commands were previously imported by the current namespace. If so, this command deletes the corresponding imported commands. In effect, this un−does the action of a namespace import command.

namespace import ?–force? ?pattern pattern ...?

    Imports commands into a namespace. Each pattern is a qualified name like foo::x or a::p*. That is, it includes the name of an exporting namespace and may have glob−style special characters in the command name at the end of the qualified name. Glob characters may not appear in a namespace name. All the commands that match a pattern string and which are currently exported from their namespace are added to the current namespace. This is done by creating a new command in the current namespace that points to the exported command in its original namespace; when the new imported command is called, it invokes the exported command. This command normally returns an error if an imported command conflicts with an existing command. However, if the –force option is given, imported commands will silently replace existing commands. The namespace import command has snapshot semantics: that is, only requested commands that are currently defined in the exporting namespace are imported. In other words, you can import only the commands that are in a namespace at the time when the namespace import command is executed. If another command is defined and exported in this namespace later on, it will not be imported.

namespace inscope namespace script ?arg ...?

    Executes a script in the context of the specified namespace. This command is not expected to be used directly by programmers; calls to it are generated implicitly when applications use namespace code commands to create callback scripts that the applications then register with, e.g., Tk widgets. The namespace inscope command is much like the namespace eval command except that the namespace must already exist, and namespace inscope appends additional args as proper list elements.
    namespace inscope ::foo $script $x $y $z is equivalent to namespace eval ::foo [concat $script
thus additional arguments will not undergo a second round of substitution, as is the case with `namespace eval`.

```
namespace origin command
```

Returns the fully-qualified name of the original command to which the imported command `command` refers. When a command is imported into a namespace, a new command is created in that namespace that points to the actual command in the exporting namespace. If a command is imported into a sequence of namespaces `a, b,...,n` where each successive namespace just imports the command from the previous namespace, this command returns the fully-qualified name of the original command in the first namespace, `a`. If `command` does not refer to an imported command, the command's own fully-qualified name is returned.

```
namespace parent ?namespace?
```

Returns the fully-qualified name of the parent namespace for namespace `namespace`. If `namespace` is not specified, the fully-qualified name of the current namespace's parent is returned.

```
namespace qualifiers string
```

Returns any leading namespace qualifiers for `string`. Qualifiers are namespace names separated by double colons (`::`). For the `string ::foo::bar::x`, this command returns `::foo::bar`, and for `::` it returns an empty string. This command is the complement of the `namespace tail` command. Note that it does not check whether the namespace names are, in fact, the names of currently defined namespaces.

```
namespace tail string
```

Returns the simple name at the end of a qualified string. Qualifiers are namespace names separated by double colons (`::`). For the `string ::foo::bar::x`, this command returns `x`, and for `::` it returns an empty string. This command is the complement of the `namespace qualifiers` command. It does not check whether the namespace names are, in fact, the names of currently defined namespaces.

```
namespace which ?--command? ?--variable? name
```

Looks up `name` as either a command or variable and returns its fully-qualified name. For example, if `name` does not exist in the current namespace but does exist in the global namespace, this command returns a fully-qualified name in the global namespace. If the command or variable does not exist, this command returns an empty string. If the variable has been created but not defined, such as with the `variable` command or through a `trace` on the variable, this command will return the fully-qualified name of the variable. If no flag is given, `name` is treated as a command name. See the section `NAME RESOLUTION` below for an explanation of the rules regarding name resolution.

WHAT IS A NAMESPACE?

A namespace is a collection of commands and variables. It encapsulates the commands and variables to ensure that they won't interfere with the commands and variables of other namespaces. Tcl has always had one such collection, which we refer to as the global namespace. The global namespace holds all global variables and commands. The `namespace eval` command lets you create new namespaces. For example,
creates a new namespace containing the variable `num` and the procedure `bump`. The commands and variables in this namespace are separate from other commands and variables in the same program. If there is a command named `bump` in the global namespace, for example, it will be different from the command `bump` in the `Counter` namespace.

Namespace variables resemble global variables in Tcl. They exist outside of the procedures in a namespace but can be accessed in a procedure via the `variable` command, as shown in the example above.

Namespaces are dynamic. You can add and delete commands and variables at any time, so you can build up the contents of a namespace over time using a series of `namespace eval` commands. For example, the following series of commands has the same effect as the namespace definition shown above:

```tcl
namespace eval Counter {
    variable num 0
    proc bump {} {
        variable num
        return [incr num]
    }
}
namespace eval Counter {
    proc test {args} {
        return $args
    }
}
namespace eval Counter {
    rename test ""
}
```

Note that the `test` procedure is added to the `Counter` namespace, and later removed via the `rename` command.

Namespaces can have other namespaces within them, so they nest hierarchically. A nested namespace is encapsulated inside its parent namespace and can not interfere with other namespaces.

**QUALIFIED NAMES**

Each namespace has a textual name such as `history` or `::safe::interp`. Since namespaces may nest, qualified names are used to refer to commands, variables, and child namespaces contained inside namespaces. Qualified names are similar to the hierarchical path names for Unix files or Tk widgets, except that `::` is used as the separator instead of `/` or `. `. The topmost or global namespace has the name `::` (i.e., an empty string), although `::` is a synonym. As an example, the name `::safe::interp::create` refers to the command `create` in the namespace `interp` that is a child of namespace `::safe`, which in turn is a child of the global namespace, `::`.

If you want to access commands and variables from another namespace, you must use some extra syntax. Names must be qualified by the namespace that contains them. From the global namespace, we might access the `Counter` procedures like this:
Counter::bump 5
Counter::Reset

We could access the current count like this:

puts "count = $Counter::num"

When one namespace contains another, you may need more than one qualifier to reach its elements. If we had a namespace Foo that contained the namespace Counter, you could invoke its bump procedure from the global namespace like this:

Foo::Counter::bump 3

You can also use qualified names when you create and rename commands. For example, you could add a procedure to the Foo namespace like this:

proc Foo::Test {args} {return $args}

And you could move the same procedure to another namespace like this:

rename Foo::Test Bar::Test

There are a few remaining points about qualified names that we should cover. Namespaces have nonempty names except for the global namespace. :: is disallowed in simple command, variable, and namespace names except as a namespace separator. Extra colons in any separator part of a qualified name are ignored; i.e. two or more colons are treated as a namespace separator. A trailing :: in a qualified variable or command name refers to the variable or command named {}. However, a trailing :: in a qualified namespace name is ignored.

NAME RESOLUTION

In general, all Tcl commands that take variable and command names support qualified names. This means you can give qualified names to such commands as set, proc, rename, and interp alias. If you provide a fully–qualified name that starts with a ::, there is no question about what command, variable, or namespace you mean. However, if the name does not start with a :: (i.e., is relative), Tcl follows a fixed rule for looking it up: Command and variable names are always resolved by looking first in the current namespace, and then in the global namespace. Namespace names, on the other hand, are always resolved by looking in only the current namespace.

In the following example,

set traceLevel 0
namespace eval Debug {
    procedure printTrace $traceLevel
}

Tcl looks for traceLevel in the namespace Debug and then in the global namespace. It looks up the command printTrace in the same way. If a variable or command name is not found in either context, the name is
undefined. To make this point absolutely clear, consider the following example:

```tcl
set traceLevel 0
namespace eval Foo {
    variable traceLevel 3

    namespace eval Debug {
        printTrace $traceLevel
    }
}
```

Here Tcl looks for `traceLevel` first in the namespace `Foo::Debug`. Since it is not found there, Tcl then looks for it in the global namespace. The variable `Foo::traceLevel` is completely ignored during the name resolution process.

You can use the `namespace which` command to clear up any question about name resolution. For example, the command:

```tcl
namespace eval Foo::Debug {namespace which −variable traceLevel}
```

returns `::traceLevel`. On the other hand, the command,

```tcl
namespace eval Foo {namespace which −variable traceLevel}
```

returns `::Foo::traceLevel`.

As mentioned above, namespace names are looked up differently than the names of variables and commands. Namespace names are always resolved in the current namespace. This means, for example, that a `namespace eval` command that creates a new namespace always creates a child of the current namespace unless the new namespace name begins with `::`.

Tcl has no access control to limit what variables, commands, or namespaces you can reference. If you provide a qualified name that resolves to an element by the name resolution rule above, you can access the element.

You can access a namespace variable from a procedure in the same namespace by using the `variable` command. Much like the `global` command, this creates a local link to the namespace variable. If necessary, it also creates the variable in the current namespace and initializes it. Note that the `global` command only creates links to variables in the global namespace. It is not necessary to use a `variable` command if you always refer to the namespace variable using an appropriate qualified name.

**IMPORTING COMMANDS**

Namespaces are often used to represent libraries. Some library commands are used so frequently that it is a nuisance to type their qualified names. For example, suppose that all of the commands in a package like BLT are contained in a namespace called `Blt`. Then you might access these commands like this:

```
Blt::graph .g −background red
```
If you use the `graph` and `table` commands frequently, you may want to access them without the `Blt::` prefix. You can do this by importing the commands into the current namespace, like this:

```tcl
namespace import Blt::*
```

This adds all exported commands from the `Blt` namespace into the current namespace context, so you can write code like this:

```tcl
graph .g -background red
table .g 0,0
```

The `namespace import` command only imports commands from a namespace that that namespace exported with a `namespace export` command.

Importing every command from a namespace is generally a bad idea since you don't know what you will get. It is better to import just the specific commands you need. For example, the command

```tcl
namespace import Blt::graph Blt::table
```

imports only the `graph` and `table` commands into the current context.

If you try to import a command that already exists, you will get an error. This prevents you from importing the same command from two different packages. But from time to time (perhaps when debugging), you may want to get around this restriction. You may want to reissue the `namespace import` command to pick up new commands that have appeared in a namespace. In that case, you can use the `−force` option, and existing commands will be silently overwritten:

```tcl
namespace import -force Blt::graph Blt::table
```

If for some reason, you want to stop using the imported commands, you can remove them with a `namespace forget` command, like this:

```tcl
namespace forget Blt::*
```

This searches the current namespace for any commands imported from `Blt`. If it finds any, it removes them. Otherwise, it does nothing. After this, the `Blt` commands must be accessed with the `Blt::` prefix.

When you delete a command from the exporting namespace like this:

```tcl
rename Blt::graph ""
```

the command is automatically removed from all namespaces that import it.
EXPORTING COMMANDS

You can export commands from a namespace like this:

```tcl
namespace eval Counter {
    namespace export bump reset
    variable Num 0
    variable Max 100

    proc bump {{by 1}} {
        variable Num
        incr Num $by
        Check
        return $Num
    }
    proc reset {} {
        variable Num
        set Num 0
    }
    proc Check {} {
        variable Num
        variable Max
        if ($Num > $Max) {
            error "too high!"
        }
    }
}
```

The procedures `bump` and `reset` are exported, so they are included when you import from the `Counter` namespace, like this:

```tcl
namespace import Counter::*
```

However, the `Check` procedure is not exported, so it is ignored by the import operation.

The `namespace import` command only imports commands that were declared as exported by their namespace. The `namespace export` command specifies what commands may be imported by other namespaces. If a `namespace import` command specifies a command that is not exported, the command is not imported.

SCOPED SCRIPTS

The `namespace code` command is the means by which a script may be packaged for evaluation in a namespace other than the one in which it was created. It is used most often to create event handlers, Tk bindings, and traces for evaluation in the global context. For instance, the following code indicates how to direct a variable trace callback into the current namespace:

```tcl
namespace eval a {
    variable b
    proc theTraceCallback { n1 n2 op } {
        # Code for the TraceCallback
    }
}
```
upvar 1 $n1 var
puts "the value of $n1 has changed to $var"
return
}
trace variable b w [namespace code theTraceCallback]
set a::b c

When executed, it prints the message:

the value of a::b has changed to c

EXAMPLES

Create a namespace containing a variable and an exported command:

namespace eval foo {
    variable bar 0
    proc grill {} {
        variable bar
        puts "called [incr bar] times"
    }
    namespace export grill
}

Call the command defined in the previous example in various ways.

# Direct call
foo::grill

# Import into current namespace, then call local alias
namespace import foo::grill
grill

Look up where the command imported in the previous example came from:

puts "grill came from [namespace which grill]"

SEE ALSO

variable

KEYWORDS

exported, internal, variable
open

NAME

open – Open a file–based or command pipeline channel

SYNOPSIS

open fileName
open fileName access
open fileName access permissions

DESCRIPTION

r
r+
w
w+
a
a+
RDONLY
WRONLY
RDWR
APPEND
CREAT
EXCL
NOCTTY
NONBLOCK
TRUNC

COMMAND PIPELINES

SERIAL COMMUNICATIONS

-mode baud,parity,data,stop
-handshake type
-queue
-timeout msec
-ttycontrol [signal boolean signal boolean ...]
-ttystatus
-xchar [xonChar xoffChar]
-pollinterval msec
-sysbuffer inSize
-sysbuffer [inSize outSize]
-lasterror

SERIAL PORT SIGNALS

TXD(output)
RXD(input)
RTS(output)
CTS(input)
DTR(output)
DSR(input)
DCD(input)
RI(input)
BREAK

ERROR CODES (Windows only)
RXOVER
TXFULL
OVERRUN
RXPARITY
FRAME
BREAK

PORTABILITY ISSUES
Windows (all versions)
Windows NT
Windows 95
Macintosh
Unix

EXAMPLE
SEE ALSO
KEYWORDS

NAME

open – Open a file–based or command pipeline channel

SYNOPSIS

open fileName
open fileName access
open fileName access permissions

DESCRIPTION

This command opens a file, serial port, or command pipeline and returns a channel identifier that may be used in future invocations of commands like read, puts, and close. If the first character of fileName is not | then the command opens a file: fileName gives the name of the file to open, and it must conform to the conventions described in the filename manual entry.

The access argument, if present, indicates the way in which the file (or command pipeline) is to be accessed. In the first form access may have any of the following values:

r
Open the file for reading only; the file must already exist. This is the default value if access is not specified.

r+

Open the file for both reading and writing; the file must already exist.

`w`
Open the file for writing only. Truncate it if it exists. If it doesn't exist, create a new file.

`w+`
Open the file for reading and writing. Truncate it if it exists. If it doesn't exist, create a new file.

`a`
Open the file for writing only. If the file doesn't exist, create a new empty file. Set the initial access position to the end of the file.

`a+`
Open the file for reading and writing. If the file doesn't exist, create a new empty file. Set the initial access position to the end of the file.

In the second form, `access` consists of a list of any of the following flags, all of which have the standard POSIX meanings. One of the flags must be either RDONLY, WRONLY or RDWR.

**RDONLY**
Open the file for reading only.

**WRONLY**
Open the file for writing only.

**RDWR**
Open the file for both reading and writing.

**APPEND**
Set the file pointer to the end of the file prior to each write.

**CREAT**
Create the file if it doesn't already exist (without this flag it is an error for the file not to exist).

**EXCL**
If `CREAT` is also specified, an error is returned if the file already exists.

**NOCTTY**
If the file is a terminal device, this flag prevents the file from becoming the controlling terminal of the process.

**NONBLOCK**
Prevents the process from blocking while opening the file, and possibly in subsequent I/O operations. The exact behavior of this flag is system– and device–dependent; its use is discouraged (it is better to use the `fconfigure` command to put a file in nonblocking mode). For details refer to your system documentation on the `open` system call’s `O_NONBLOCK` flag.

**TRUNC**
If the file exists it is truncated to zero length.

If a new file is created as part of opening it, `permissions` (an integer) is used to set the permissions for the new file in conjunction with the process's file mode creation mask. `Permissions` defaults to 0666.

Note that if you are going to be reading or writing binary data from the channel created by this command, you should use the `fconfigure` command to change the `-translation` option of the channel to `binary` before transferring any binary data. This is in contrast to the ``b`` character passed as part of the equivalent of the `access` parameter to some versions of the C library `fopen()` function.
COMMAND PIPELINES

If the first character of fileName is `|' then the remaining characters of fileName are treated as a list of arguments that describe a command pipeline to invoke, in the same style as the arguments for exec. In this case, the channel identifier returned by open may be used to write to the command's input pipe or read from its output pipe, depending on the value of access. If write–only access is used (e.g. access is w), then standard output for the pipeline is directed to the current standard output unless overridden by the command. If read–only access is used (e.g. access is r), standard input for the pipeline is taken from the current standard input unless overridden by the command. The id of the spawned process is accessible through the pid command, using the channel id returned by open as argument.

If the command (or one of the commands) executed in the command pipeline returns an error (according to the definition in exec), a Tcl error is generated when close is called on the channel (similar to the close command.)

It is often useful to use the fileevent command with pipelines so other processing may happen at the same time as running the command in the background.

SERIAL COMMUNICATIONS

If fileName refers to a serial port, then the specified serial port is opened and initialized in a platform–dependent manner. Acceptable values for the fileName to use to open a serial port are described in the PORTABILITY ISSUES section.

The fconfigure command can be used to query and set additional configuration options specific to serial ports (where supported):

−mode baud,parity,data,stop
  This option is a set of 4 comma–separated values: the baud rate, parity, number of data bits, and number of stop bits for this serial port. The baud rate is a simple integer that specifies the connection speed. Parity is one of the following letters: n, o, e, m, s; respectively signifying the parity options of `none', `odd', `even', `mark', or `space'. Data is the number of data bits and should be an integer from 5 to 8, while stop is the number of stop bits and should be the integer 1 or 2.

−handshake type
  (Windows and Unix). This option is used to setup automatic handshake control. Note that not all handshake types maybe supported by your operating system. The type parameter is case–independent.

If type is none then any handshake is switched off. rtscts activates hardware handshake. Hardware handshake signals are described below. For software handshake xonxoff the handshake characters can be redefined with −xchar. An additional hardware handshake dtrd sr is available only under Windows. There is no default handshake configuration, the initial value depends on your operating system settings. The −handshake option cannot be queried.

−queue
  (Windows and Unix). The −queue option can only be queried. It returns a list of two integers representing the current number of bytes in the input and output queue respectively.
−timeout msec
(Windows and Unix). This option is used to set the timeout for blocking read operations. It specifies the maximum interval between the reception of two bytes in milliseconds. For Unix systems the granularity is 100 milliseconds. The −timeout option does not affect write operations or nonblocking reads. This option cannot be queried.

−ttycontrol [signal boolean signal boolean ...]
(Windows and Unix). This option is used to setup the handshake output lines (see below) permanently or to send a BREAK over the serial line. The signal names are case-independent. \{RTS 1 DTR 0\} sets the RTS output to high and the DTR output to low. The BREAK condition (see below) is enabled and disabled with \{BREAK 1\} and \{BREAK 0\} respectively. It's not a good idea to change the RTS (or DTR) signal with active hardware handshake rtscts (or dtrdsr). The result is unpredictable. The −ttycontrol option cannot be queried.

−ttystatus
(Windows and Unix). The −ttystatus option can only be queried. It returns the current modem status and handshake input signals (see below). The result is a list of signal,value pairs with a fixed order, e.g. \{CTS 1 DSR 0 RING 1 DCD 0\}. The signal names are returned upper case.

−xchar \{xonChar xoffChar\}
(Windows and Unix). This option is used to query or change the software handshake characters. Normally the operating system default should be DC1 (0x11) and DC3 (0x13) representing the ASCII standard XON and XOFF characters.

−pollinterval msec
(Windows only). This option is used to set the maximum time between polling for fileevents. This affects the time interval between checking for events throughout the Tcl interpreter (the smallest value always wins). Use this option only if you want to poll the serial port more or less often than 10 msec (the default).

−sysbuffer inSize
−sysbuffer \{inSize outSize\}
(Windows only). This option is used to change the size of Windows system buffers for a serial channel. Especially at higher communication rates the default input buffer size of 4096 bytes can overrun for latent systems. The first form specifies the input buffer size, in the second form both input and output buffers are defined.

−lasterror
(Windows only). This option is query only. In case of a serial communication error, read or puts returns a general Tcl file I/O error. fconfigure −lasterror can be called to get a list of error details. See below for an explanation of the various error codes.

SERIAL PORT SIGNALS

RS–232 is the most commonly used standard electrical interface for serial communications. A negative voltage (−3V..−12V) define a mark (on=1) bit and a positive voltage (+3..+12V) define a space (off=0) bit (RS–232C). The following signals are specified for incoming and outgoing data, status lines and handshaking. Here we are using the terms workstation for your computer and modem for the external device, because some signal names (DCD, RI) come from modems. Of course your external device may use these signal lines for other purposes.
**TXD(output)**

**Transmitted Data:** Outgoing serial data.

**RXD(input)**

**Received Data:** Incoming serial data.

**RTS(output)**

**Request To Send:** This hardware handshake line informs the modem that your workstation is ready to receive data. Your workstation may automatically reset this signal to indicate that the input buffer is full.

**CTS(input)**

**Clear To Send:** The complement to RTS. Indicates that the modem is ready to receive data.

**DTR(output)**

**Data Terminal Ready:** This signal tells the modem that the workstation is ready to establish a link. DTR is often enabled automatically whenever a serial port is opened.

**DSR(input)**

**Data Set Ready:** The complement to DTR. Tells the workstation that the modem is ready to establish a link.

**DCD(input)**

**Data Carrier Detect:** This line becomes active when a modem detects a "Carrier" signal.

**RI(input)**

**Ring Indicator:** Goes active when the modem detects an incoming call.

**BREAK**

A BREAK condition is not a hardware signal line, but a logical zero on the TXD or RXD lines for a long period of time, usually 250 to 500 milliseconds. Normally a receive or transmit data signal stays at the mark (on=1) voltage until the next character is transferred. A BREAK is sometimes used to reset the communications line or change the operating mode of communications hardware.

---

**ERROR CODES (Windows only)**

A lot of different errors may occur during serial read operations or during event polling in background. The external device may have been switched off, the data lines may be noisy, system buffers may overrun or your mode settings may be wrong. That's why a reliable software should always catch serial read operations. In cases of an error Tcl returns a general file I/O error. Then fconfigure –lasterror may help to locate the problem. The following error codes may be returned.

**RXOVER**

Windows input buffer overrun. The data comes faster than your scripts reads it or your system is overloaded. Use fconfigure –sysbuffer to avoid a temporary bottleneck and/or make your script faster.

**TXFULL**

Windows output buffer overrun. Complement to RXOVER. This error should practically not happen, because Tcl cares about the output buffer status.

**OVERRUN**

UART buffer overrun (hardware) with data lost. The data comes faster than the system driver receives it. Check your advanced serial port settings to enable the FIFO (16550) buffer and/or setup a lower(1) interrupt threshold value.
**RXPARITY**
A parity error has been detected by your UART. Wrong parity settings with `fconfigure –mode` or a noisy data line (RXD) may cause this error.

**FRAME**
A stop–bit error has been detected by your UART. Wrong mode settings with `fconfigure –mode` or a noisy data line (RXD) may cause this error.

**BREAK**
A BREAK condition has been detected by your UART (see above).

**PORTABILITY ISSUES**

**Windows (all versions)**
Valid values for `fileName` to open a serial port are of the form `comX:`, where `X` is a number, generally from 1 to 4. This notation only works for serial ports from 1 to 9, if the system happens to have more than four. An attempt to open a serial port that does not exist or has a number greater than 9 will fail. An alternate form of opening serial ports is to use the filename `\comX`, where `X` is any number that corresponds to a serial port; please note that this method is considerably slower on Windows 95 and Windows 98.

**Windows NT**
When running Tcl interactively, there may be some strange interactions between the real console, if one is present, and a command pipeline that uses standard input or output. If a command pipeline is opened for reading, some of the lines entered at the console will be sent to the command pipeline and some will be sent to the Tcl evaluator. If a command pipeline is opened for writing, keystrokes entered into the console are not visible until the pipe is closed. This behavior occurs whether the command pipeline is executing 16–bit or 32–bit applications. These problems only occur because both Tcl and the child application are competing for the console at the same time. If the command pipeline is started from a script, so that Tcl is not accessing the console, or if the command pipeline does not use standard input or output, but is redirected from or to a file, then the above problems do not occur.

**Windows 95**
A command pipeline that executes a 16–bit DOS application cannot be opened for both reading and writing, since 16–bit DOS applications that receive standard input from a pipe and send standard output to a pipe run synchronously. Command pipelines that do not execute 16–bit DOS applications run asynchronously and can be opened for both reading and writing.

When running Tcl interactively, there may be some strange interactions between the real console, if one is present, and a command pipeline that uses standard input or output. If a command pipeline is opened for reading from a 32–bit application, some of the keystrokes entered at the console will be sent to the command pipeline and some will be sent to the Tcl evaluator. If a command pipeline is opened for writing to a 32–bit application, no output is visible on the console until the pipe is closed. These problems only occur because both Tcl and the child application are competing for the console at the same time. If the command pipeline is started from a script, so that Tcl is not accessing the console, or if the command pipeline does not use standard input or output, but is redirected from or to a file, then the above problems do not occur.
Whether or not Tcl is running interactively, if a command pipeline is opened for reading from a 16–bit DOS application, the call to `open` will not return until end–of–file has been received from the command pipeline's standard output. If a command pipeline is opened for writing to a 16–bit DOS application, no data will be sent to the command pipeline's standard output until the pipe is actually closed. This problem occurs because 16–bit DOS applications are run synchronously, as described above.

**Macintosh**

Opening a serial port is not currently implemented under Macintosh.

Opening a command pipeline is not supported under Macintosh, since applications do not support the concept of standard input or output.

**Unix**

Valid values for `fileName` to open a serial port are generally of the form `/dev/ttyX`, where `X` is `a` or `b`, but the name of any pseudo–file that maps to a serial port may be used. Advanced configuration options are only supported for serial ports when Tcl is built to use the POSIX serial interface.

When running Tcl interactively, there may be some strange interactions between the console, if one is present, and a command pipeline that uses standard input. If a command pipeline is opened for reading, some of the lines entered at the console will be sent to the command pipeline and some will be sent to the Tcl evaluator. This problem only occurs because both Tcl and the child application are competing for the console at the same time. If the command pipeline is started from a script, so that Tcl is not accessing the console, or if the command pipeline does not use standard input, but is redirected from a file, then the above problem does not occur.

See the PORTABILITY ISSUES section of the `exec` command for additional information not specific to command pipelines about executing applications on the various platforms.

**EXAMPLE**

Open a command pipeline and catch any errors:

```tcl
set fl [open "| ls this_file_does_not_exist"]
set data [read $fl]
if {[catch {close $fl} err]} {
    puts "ls command failed: $err"
}
```

**SEE ALSO**

`file`, `close`, `filename`, `fconfigure`, `gets`, `read`, `puts`, `exec`, `pid`, `fopen`

**KEYWORDS**

`access mode`, `append`, `create`, `file`, `non–blocking`, `open`, `permissions`, `pipeline`, `process`, `serial`
NAME

package – Facilities for package loading and version control

SYNOPSIS

package forget ?package package ...?
package ifneeded package version ?script?
package names
package present ?−exact? package ?version?
package provide package ?version?
package require ?−exact? package ?version?
package unknown ?command?
package vcompare version1 version2
package versions package
package vsatisfies version1 version2

DESCRIPTION

package forget ?package package ...?
package ifneeded package version ?script?
package names
package present ?−exact? package ?version?
package provide package ?version?
package require ?−exact? package ?version?
package unknown ?command?
package vcompare version1 version2
package versions package
package vsatisfies version1 version2

VERSION NUMBERS

PACKAGE INDICES

EXAMPLES

SEE ALSO

KEYWORDS
package provide package ?version?
package require ?−exact? package ?version?
package unknown ?command?
package vcompare version1 version2
package versions package
package vsatisfies version1 version2

DESCRIPTION

This command keeps a simple database of the packages available for use by the current interpreter and how to load them into the interpreter. It supports multiple versions of each package and arranges for the correct version of a package to be loaded based on what is needed by the application. This command also detects and reports version clashes. Typically, only the package require and package provide commands are invoked in normal Tcl scripts; the other commands are used primarily by system scripts that maintain the package database.

The behavior of the package command is determined by its first argument. The following forms are permitted:

package forget ?package package ...?
   Removes all information about each specified package from this interpreter, including information provided by both package ifneeded and package provide.

package ifneeded package version ?script?
   This command typically appears only in system configuration scripts to set up the package database. It indicates that a particular version of a particular package is available if needed, and that the package can be added to the interpreter by executing script. The script is saved in a database for use by subsequent package require commands; typically, script sets up auto-loading for the commands in the package (or calls load and/or source directly), then invokes package provide to indicate that the package is present. There may be information in the database for several different versions of a single package. If the database already contains information for package and version, the new script replaces the existing one. If the script argument is omitted, the current script for version version of package package is returned, or an empty string if no package ifneeded command has been invoked for this package and version.

package names
   Returns a list of the names of all packages in the interpreter for which a version has been provided (via package provide) or for which a package ifneeded script is available. The order of elements in the list is arbitrary.

package present ?−exact? package ?version?
   This command is equivalent to package require except that it does not try and load the package if it is not already loaded.

package provide package ?version?
   This command is invoked to indicate that version version of package package is now present in the interpreter. It is typically invoked once as part of an ifneeded script, and again by the package itself when it is finally loaded. An error occurs if a different version of package has been provided by a previous package provide command. If the version argument is omitted, then the command returns
the version number that is currently provided, or an empty string if no package provide command has been invoked for package in this interpreter.

**package require** ?−exact? package ?version?

This command is typically invoked by Tcl code that wishes to use a particular version of a particular package. The arguments indicate which package is wanted, and the command ensures that a suitable version of the package is loaded into the interpreter. If the command succeeds, it returns the version number that is loaded; otherwise it generates an error. If both the −exact switch and the version argument are specified then only the given version is acceptable. If −exact is omitted but version is specified, then versions later than version are also acceptable as long as they have the same major version number as version. If both −exact and version are omitted then any version whatsoever is acceptable. If a version of package has already been provided (by invoking the package provide command), then its version number must satisfy the criteria given by −exact and version and the command returns immediately. Otherwise, the command searches the database of information provided by previous package ifneeded commands to see if an acceptable version of the package is available. If so, the script for the highest acceptable version number is evaluated in the global namespace; it must do whatever is necessary to load the package, including calling package provide for the package. If the package ifneeded database does not contain an acceptable version of the package and a package unknown command has been specified for the interpreter then that command is evaluated in the global namespace; when it completes, Tcl checks again to see if the package is now provided or if there is a package ifneeded script for it. If all of these steps fail to provide an acceptable version of the package, then the command returns an error.

**package unknown** ?command?

This command supplies a "last resort" command to invoke during package require if no suitable version of a package can be found in the package ifneeded database. If the command argument is supplied, it contains the first part of a command; when the command is invoked during a package require command, Tcl appends two additional arguments giving the desired package name and version. For example, if command is foo bar and later the command package require test 2.4 is invoked, then Tcl will execute the command foo bar test 2.4 to load the package. If no version number is supplied to the package require command, then the version argument for the invoked command will be an empty string. If the package unknown command is invoked without a command argument, then the current package unknown script is returned, or an empty string if there is none. If command is specified as an empty string, then the current package unknown script is removed, if there is one.

**package vcompare** version1 version2

Compares the two version numbers given by version1 and version2. Returns −1 if version1 is an earlier version than version2, 0 if they are equal, and 1 if version1 is later than version2.

**package versions** package

Returns a list of all the version numbers of package for which information has been provided by package ifneeded commands.

**package vsatisfies** version1 version2

Returns 1 if scripts written for version2 will work unchanged with version1 (i.e. version1 is equal to or greater than version2 and they both have the same major version number), 0 otherwise.
VERSION NUMBERS

Version numbers consist of one or more decimal numbers separated by dots, such as 2 or 1.162 or 3.1.13.1. The first number is called the major version number. Larger numbers correspond to later versions of a package, with leftmost numbers having greater significance. For example, version 2.1 is later than 1.3 and version 3.4.6 is later than 3.3.5. Missing fields are equivalent to zeroes: version 1.3 is the same as version 1.3.0 and 1.3.0.0, so it is earlier than 1.3.1 or 1.3.0.2. A later version number is assumed to be upwards compatible with an earlier version number as long as both versions have the same major version number. For example, Tcl scripts written for version 2.3 of a package should work unchanged under versions 2.3.2, 2.4, and 2.5.1. Changes in the major version number signify incompatible changes: if code is written to use version 2.1 of a package, it is not guaranteed to work unmodified with either version 1.7.3 or version 3.1.

PACKAGE INDICES

The recommended way to use packages in Tcl is to invoke `package require` and `package provide` commands in scripts, and use the procedure `pkg_mkIndex` to create package index files. Once you've done this, packages will be loaded automatically in response to `package require` commands. See the documentation for `pkg_mkIndex` for details.

EXAMPLES

To state that a Tcl script requires the Tk and http packages, put this at the top of the script:

```tcl
package require Tk
package require http
```

To test to see if the Snack package is available and load if it is (often useful for optional enhancements to programs where the loss of the functionality is not critical) do this:

```tcl
if { ![catch {package require Snack}]} {
   # We have the package, configure the app to use it
} else {
   # Set up a dummy interface to work around the absence
}
```

When writing a package implementation, you should put the following at the bottom of your library script so it is only called once the package has been successfully set up:

```tcl
package provide foobar 1.0
```

SEE ALSO

`msgcat`, `packagens`, `pkgMkIndex`
KEYWORDS

package, version

Copyright ' 1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
pid

NAME

pid – Retrieve process identifiers

SYNOPSIS

pid ?fileId?

DESCRIPTION

If the fileId argument is given then it should normally refer to a process pipeline created with the open command. In this case the pid command will return a list whose elements are the process identifiers of all the processes in the pipeline, in order. The list will be empty if fileId refers to an open file that isn’t a process pipeline. If no fileId argument is given then pid returns the process identifier of the current process. All process identifiers are returned as decimal strings.

EXAMPLE

Print process information about the processes in a pipeline using the SysV ps program before reading the output of that pipeline:

```tcl
set pipeline [open "| zcat somefile.gz | grep foobar | sort -u"]
# Print process information
exec ps -fp [pid $pipeline] >@stdout
# Print a separator and then the output of the pipeline
puts [string repeat - 70]
puts [read $pipeline]
close $pipeline
```

SEE ALSO

exec, open

KEYWORDS

file, pipeline, process identifier

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

pkg::create – Construct an appropriate package ifneeded command for a given package specification

SYNOPSIS

::pkg::create −name packageName −version packageVersion ?−load filespec? ... ?−source filespec? ...

DESCRIPTION

::pkg::create is a utility procedure that is part of the standard Tcl library. It is used to create an appropriate package ifneeded command for a given package specification. It can be used to construct a pkgIndex.tcl file for use with the package mechanism.

OPTIONS

The parameters supported are:

−name packageName
   This parameter specifies the name of the package. It is required.

−version packageVersion
   This parameter specifies the version of the package. It is required.

−load filespec
   This parameter specifies a binary library that must be loaded with the load command. filespec is a list with two elements. The first element is the name of the file to load. The second, optional element is a
list of commands supplied by loading that file. If the list of procedures is empty or omitted, 
::pkg::create will set up the library for direct loading (see pkg_mkIndex). Any number of –load 
parameters may be specified.

–source filespec

This parameter is similar to the –load parameter, except that it specifies a Tcl library that must be 
loaded with the source command. Any number of –source parameters may be specified.

At least one –load or –source parameter must be given.

SEE ALSO

package

KEYWORDS

auto−load, index, package, version

Copyright ' 1995−1997 Roger E. Critchlow Jr.
pkgMkIndex

NAME

pkg_mkIndex – Build an index for automatic loading of packages

SYNOPSIS


DESCRIPTION

OPTIONS

–direct
–lazy
–load pkgPat
–verbose

PACKAGES AND THE AUTO-LOADER

HOW IT WORKS

DIRECT LOADING

COMPLEX CASES

SEE ALSO

KEYWORDS

NAME

pkg_mkIndex – Build an index for automatic loading of packages

SYNOPSIS


DESCRIPTION

Pkg_mkIndex is a utility procedure that is part of the standard Tcl library. It is used to create index files that allow packages to be loaded automatically when package require commands are executed. To use pkg_mkIndex, follow these steps:

[1] Create the package(s). Each package may consist of one or more Tcl script files or binary files. Binary files must be suitable for loading with the load command with a single argument; for example, if the file is test.so it must be possible to load this file with the command load test.so. Each script file must contain a package provide command to declare the package and version number, and each binary file must contain a call to Tcl_PkgProvide.

[2]
Create the index by invoking `pkg_mkIndex`. The `dir` argument gives the name of a directory and each `pattern` argument is a `glob`-style pattern that selects script or binary files in `dir`. The default pattern is `*.tcl` and `*[info sharedlibextension]`. `Pkg_mkIndex` will create a file `pkgIndex.tcl` in `dir` with package information about all the files given by the `pattern` arguments. It does this by loading each file into a slave interpreter and seeing what packages and new commands appear (this is why it is essential to have `package provide` commands or `Tcl_PkgProvide` calls in the files, as described above). If you have a package split among scripts and binary files, or if you have dependencies among files, you may have to use the `−load` option or adjust the order in which `pkg_mkIndex` processes the files. See COMPLEX CASES below.

Install the package as a subdirectory of one of the directories given by the `tcl_pkgPath` variable. If `$tcl_pkgPath` contains more than one directory, machine-dependent packages (e.g., those that contain binary shared libraries) should normally be installed under the first directory and machine-independent packages (e.g., those that contain only Tcl scripts) should be installed under the second directory. The subdirectory should include the package's script and/or binary files as well as the `pkgIndex.tcl` file. As long as the package is installed as a subdirectory of a directory in `$tcl_pkgPath` it will automatically be found during `package require` commands.

If you install the package anywhere else, then you must ensure that the directory containing the package is in the `auto_path` global variable or an immediate subdirectory of one of the directories in `auto_path`. `Auto_path` contains a list of directories that are searched by both the auto-loader and the package loader; by default it includes `$tcl_pkgPath`. The package loader also checks all of the subdirectories of the directories in `auto_path`. You can add a directory to `auto_path` explicitly in your application, or you can add the directory to your `TCLLIBPATH` environment variable: if this environment variable is present, Tcl initializes `auto_path` from it during application startup.

Once the above steps have been taken, all you need to do to use a package is to invoke `package require`. For example, if versions 2.1, 2.3, and 3.1 of package `Test` have been indexed by `pkg_mkIndex`, the command `package require Test` will make version 3.1 available and the command `package require −exact Test 2.1` will make version 2.1 available. There may be many versions of a package in the various index files in `auto_path`, but only one will actually be loaded in a given interpreter, based on the first call to `package require`. Different versions of a package may be loaded in different interpreters.

**OPTIONS**

The optional switches are:

−`direct`

The generated index will implement direct loading of the package upon `package require`. This is the default.

−`lazy`

The generated index will manage to delay loading the package until the use of one of the commands provided by the package, instead of loading it immediately upon `package require`.

−`load pkgPat`

The index process will pre-load any packages that exist in the current interpreter and match `pkgPat`
into the slave interpreter used to generate the index. The pattern match uses string match rules, but without making case distinctions. See COMPLEX CASES below.

--verbose
Generate output during the indexing process. Output is via the \texttt{tclLog} procedure, which by default prints to stderr.

--
End of the flags, in case \texttt{dir} begins with a dash.

\textbf{PACKAGES AND THE AUTO−LOADER}

The package management facilities overlap somewhat with the auto−loader, in that both arrange for files to be loaded on−demand. However, package management is a higher−level mechanism that uses the auto−loader for the last step in the loading process. It is generally better to index a package with \texttt{pkg\_mkIndex} rather than \texttt{auto\_mkindex} because the package mechanism provides version control: several versions of a package can be made available in the index files, with different applications using different versions based on \texttt{package require} commands. In contrast, \texttt{auto\_mkindex} does not understand versions so it can only handle a single version of each package. It is probably not a good idea to index a given package with both \texttt{pkg\_mkIndex} and \texttt{auto\_mkindex}. If you use \texttt{pkg\_mkIndex} to index a package, its commands cannot be invoked until \texttt{package require} has been used to select a version; in contrast, packages indexed with \texttt{auto\_mkindex} can be used immediately since there is no version control.

\textbf{HOW IT WORKS}

\texttt{Pkg\_mkIndex} depends on the \texttt{package unknown} command, the \texttt{package ifneeded} command, and the auto−loader. The first time a \texttt{package require} command is invoked, the \texttt{package unknown} script is invoked. This is set by Tcl initialization to a script that evaluates all of the \texttt{pkgIndex.tcl} files in the \texttt{auto\_path}. The \texttt{pkgIndex.tcl} files contain \texttt{package ifneeded} commands for each version of each available package; these commands invoke \texttt{package provide} commands to announce the availability of the package, and they setup auto−loader information to load the files of the package. If the \texttt{−lazy} flag was provided when the \texttt{pkgIndex.tcl} was generated, a given file of a given version of a given package isn't actually loaded until the first time one of its commands is invoked. Thus, after invoking \texttt{package require} you may not see the package's commands in the interpreter, but you will be able to invoke the commands and they will be auto−loaded.

\textbf{DIRECT LOADING}

Some packages, for instance packages which use namespaces and export commands or those which require special initialization, might select that their package files be loaded immediately upon \texttt{package require} instead of delaying the actual loading to the first use of one of the package's command. This is the default mode when generating the package index. It can be overridden by specifying the \texttt{−lazy} argument.

\textbf{COMPLEX CASES}

Most complex cases of dependencies among scripts and binary files, and packages being split among scripts and binary files are handled OK. However, you may have to adjust the order in which files are processed by
pkg_mkIndex. These issues are described in detail below.

If each script or file contains one package, and packages are only contained in one file, then things are easy. You simply specify all files to be indexed in any order with some glob patterns.

In general, it is OK for scripts to have dependencies on other packages. If scripts contain package require commands, these are stubbed out in the interpreter used to process the scripts, so these do not cause problems. If scripts call into other packages in global code, these calls are handled by a stub unknown command. However, if scripts make variable references to other package's variables in global code, these will cause errors. That is also bad coding style.

If binary files have dependencies on other packages, things can become tricky because it is not possible to stub out C−level APIs such as Tcl_PkgRequire API when loading a binary file. For example, suppose the BLT package requires Tk, and expresses this with a call to Tcl_PkgRequire in its Blt_Init routine. To support this, you must run pkg_mkIndex in an interpreter that has Tk loaded. You can achieve this with the −load pkgPat option. If you specify this option, pkg_mkIndex will load any packages listed by info loaded and that match pkgPat into the interpreter used to process files. In most cases this will satisfy the Tcl_PkgRequire calls made by binary files.

If you are indexing two binary files and one depends on the other, you should specify the one that has dependencies last. This way the one without dependencies will get loaded and indexed, and then the package it provides will be available when the second file is processed. You may also need to load the first package into the temporary interpreter used to create the index by using the −load flag; it won't hurt to specify package patterns that are not yet loaded.

If you have a package that is split across scripts and a binary file, then you should avoid the −load flag. The problem is that if you load a package before computing the index it masks any other files that provide part of the same package. If you must use −load, then you must specify the scripts first; otherwise the package loaded from the binary file may mask the package defined by the scripts.

SEE ALSO

package

KEYWORDS

auto−load, index, package, version

Copyright © 1996 Sun Microsystems, Inc.
Copyright © 1995-1997 Roger E. Critchlow Jr.
proc

NAME

proc – Create a Tcl procedure

SYNOPSIS

proc name args body

DESCRIPTION

The proc command creates a new Tcl procedure named name, replacing any existing command or procedure there may have been by that name. Whenever the new command is invoked, the contents of body will be executed by the Tcl interpreter. Normally, name is unqualified (does not include the names of any containing namespaces), and the new procedure is created in the current namespace. If name includes any namespace qualifiers, the procedure is created in the specified namespace. Args specifies the formal arguments to the procedure. It consists of a list, possibly empty, each of whose elements specifies one argument. Each argument specifier is also a list with either one or two fields. If there is only a single field in the specifier then it is the name of the argument; if there are two fields, then the first is the argument name and the second is its default value.

When name is invoked a local variable will be created for each of the formal arguments to the procedure; its value will be the value of corresponding argument in the invoking command or the argument's default value. Arguments with default values need not be specified in a procedure invocation. However, there must be enough actual arguments for all the formal arguments that don't have defaults, and there must not be any extra actual arguments. There is one special case to permit procedures with variable numbers of arguments. If the last formal argument has the name args, then a call to the procedure may contain more actual arguments than the procedure has formals. In this case, all of the actual arguments starting at the one that would be assigned to args are combined into a list (as if the list command had been used); this combined value is assigned to the local variable args.

When body is being executed, variable names normally refer to local variables, which are created automatically when referenced and deleted when the procedure returns. One local variable is automatically created for each of the procedure's arguments. Global variables can only be accessed by invoking the global command or the upvar command. Namespace variables can only be accessed by invoking the variable command or the upvar command.

The proc command returns an empty string. When a procedure is invoked, the procedure's return value is the value specified in a return command. If the procedure doesn't execute an explicit return, then its return value is the value of the last command executed in the procedure's body. If an error occurs while executing the procedure body, then the procedure—as-a-whole will return that same error.
EXAMPLES

This is a procedure that accepts arbitrarily many arguments and prints them out, one by one.

```tcl
proc printArguments args {
    foreach arg $args {
        puts $arg
    }
}
```

This procedure is a bit like the `incr` command, except it multiplies the contents of the named variable by the value, which defaults to 2:

```tcl
proc mult {varName {multiplier 2}} {
    upvar 1 $varName var
    set var [expr {$var * $multiplier}]
}
```

SEE ALSO

`info`, `unknown`

KEYWORDS

argument, procedure

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
puts

NAME

puts – Write to a channel

SYNOPSIS

puts ?–nonewline? ?channelId? string

DESCRIPTION

Writes the characters given by string to the channel given by channelId.

ChannelId must be an identifier for an open channel such as a Tcl standard channel (stdout or stderr), the return value from an invocation of open or socket, or the result of a channel creation command provided by a Tcl extension. The channel must have been opened for output.

If no channelId is specified then it defaults to stdout. Puts normally outputs a newline character after string, but this feature may be suppressed by specifying the –nonewline switch.

Newline characters in the output are translated by puts to platform-specific end-of-line sequences according to the current value of the –translation option for the channel (for example, on PCs newlines are normally replaced with carriage-return-linefeed sequences; on Macintoshes newlines are normally replaced with carriage-returns). See the fconfigure manual entry for a discussion on ways in which fconfigure will alter output.

Tcl buffers output internally, so characters written with puts may not appear immediately on the output file or device; Tcl will normally delay output until the buffer is full or the channel is closed. You can force output to appear immediately with the flush command.

When the output buffer fills up, the puts command will normally block until all the buffered data has been accepted for output by the operating system. If channelId is in nonblocking mode then the puts command will not block even if the operating system cannot accept the data. Instead, Tcl continues to buffer the data and writes it in the background as fast as the underlying file or device can accept it. The application must use the Tcl event loop for nonblocking output to work; otherwise Tcl never finds out that the file or device is ready for more output data. It is possible for an arbitrarily large amount of data to be buffered for a channel in nonblocking mode, which could consume a large amount of memory. To avoid wasting memory, nonblocking I/O should normally be used in an event-driven fashion with the fileevent command (don't invoke puts unless you have recently been notified via a file event that the channel is ready for more output data).
EXAMPLES

Write a short message to the console (or wherever stdout is directed):

```tcl```
puts "Hello, World!"
```tcl```

Print a message in several parts:

```tcl```
puts -nonewline "Hello, 
puts "World!"
```tcl```

Print a message to the standard error channel:

```tcl```
puts stderr "Hello, World!"
```tcl```

Append a log message to a file:

```tcl```
set chan [open my.log a]
set timestamp [clock format [clock seconds]]
puts $chan "$timestamp − Hello, World!"
close $chan
```tcl```

SEE ALSO

file, fileevent, Tcl_StandardChannels

KEYWORDS

channel, newline, output, write
pwd

NAME

pwd – Return the absolute path of the current working directory

SYNOPSIS

pwd

DESCRIPTION

Returns the absolute path name of the current working directory.

EXAMPLE

Sometimes it is useful to change to a known directory when running some external command using `exec`, but it is important to keep the application usually running in the directory that it was started in (unless the user specifies otherwise) since that minimises user confusion. The way to do this is to save the current directory while the external command is being run:

```
set tarFile [file normalize somefile.tar]
set savedDir [pwd]
cd /tmp
exec tar -xf $tarFile
cd $savedDir
```

SEE ALSO

file, cd, glob, filename

KEYWORDS

working directory
NAME

re_syntax – Syntax of Tcl regular expressions.

DESCRIPTION

DIFFERENT FLAVORS OF REs

REGULAR EXPRESSION SYNTAX

*  
+  
?  
{m}  
{m,n}  
{m,n}  
*? +? ?? {m}?? {m,n}?  
(re)  
(?::re)  
@  
(?:)  
[chars]

\  
\a  
\b  
\c  
\f  
\n  
\r  
\t  
\u  
\wxyz  
\Ustuvwxyz  
\w  
\\\shhh
re_syntax – Syntax of Tcl regular expressions.

DESCRIPTION

A regular expression describes strings of characters. It's a pattern that matches certain strings and doesn't match others.
DIFFERENT FLAVORS OF REs

Regular expressions (``RE''s), as defined by POSIX, come in two flavors: extended REs (``ERE's'') and basic REs (``BRE's''). EREs are roughly those of the traditional egrep, while BREs are roughly those of the traditional ed. This implementation adds a third flavor, advanced REs (``ARE's''), basically EREs with some significant extensions.

This manual page primarily describes AREs. BREs mostly exist for backward compatibility in some old programs; they will be discussed at the end. POSIX EREs are almost an exact subset of AREs. Features of AREs that are not present in EREs will be indicated.

REGULAR EXPRESSION SYNTAX

Tcl regular expressions are implemented using the package written by Henry Spencer, based on the 1003.2 spec and some (not quite all) of the Perl5 extensions (thanks, Henry!). Much of the description of regular expressions below is copied verbatim from his manual entry.

An ARE is one or more branches, separated by `|', matching anything that matches any of the branches.

A branch is zero or more constraints or quantified atoms, concatenated. It matches a match for the first, followed by a match for the second, etc; an empty branch matches the empty string.

A quantified atom is an atom possibly followed by a single quantifier. Without a quantifier, it matches a match for the atom. The quantifiers, and what a so–quantified atom matches, are:

* a sequence of 0 or more matches of the atom
+ a sequence of 1 or more matches of the atom
? a sequence of 0 or 1 matches of the atom
{m} a sequence of exactly m matches of the atom
{m,} a sequence of m or more matches of the atom
{m,n} a sequence of m through n (inclusive) matches of the atom; m may not exceed n
*? +? ?? {m}? {m,}? {m,n}?
   non–greedy quantifiers, which match the same possibilities, but prefer the smallest number rather than the largest number of matches (see MATCHING)

The forms using { and } are known as bounds. The numbers m and n are unsigned decimal integers with permissible values from 0 to 255 inclusive.

An atom is one of:
(re)
  (where re is any regular expression) matches a match for re, with the match noted for possible reporting

(?::re)
  as previous, but does no reporting (a "non-capturing" set of parentheses)

() matches an empty string, noted for possible reporting

(?:) matches an empty string, without reporting

[chars]
  a bracket expression, matching any one of the chars (see BRACKET EXPRESSIONS for more detail)

\(
  \text{matches any single character}
\)

\k
  (where k is a non-alphanumeric character) matches that character taken as an ordinary character, e.g. \ matches a backslash character

\c
  where c is alphanumeric (possibly followed by other characters), an escape (AREs only), see ESCAPES below

{\text{when followed by a character other than a digit, matches the left-brace character \'}\text{; when followed by a digit, it is the beginning of a bound (see above)}

x
  where x is a single character with no other significance, matches that character.

A constraint matches an empty string when specific conditions are met. A constraint may not be followed by a quantifier. The simple constraints are as follows; some more constraints are described later, under ESCAPES.

\`
  \text{matches at the beginning of a line}
\`

\$
  \text{matches at the end of a line}

(\?=re)
  positive lookahead (AREs only), matches at any point where a substring matching re begins

(\?!re)
  negative lookahead (AREs only), matches at any point where no substring matching re begins

The lookahead constraints may not contain back references (see later), and all parentheses within them are considered non-capturing.

An RE may not end with `\'.

DIFFERENT FLAVORS OF REs
BRACKET EXPRESSIONS

A bracket expression is a list of characters enclosed in `[ ]`. It normally matches any single character from the list (but see below). If the list begins with `^`, it matches any single character (but see below) not from the rest of the list.

If two characters in the list are separated by `−`, this is shorthand for the full range of characters between those two (inclusive) in the collating sequence, e.g. [0–9] in ASCII matches any decimal digit. Two ranges may not share an endpoint, so e.g. a–c–e is illegal. Ranges are very collating-sequence-dependent, and portable programs should avoid relying on them.

To include a literal ] or − in the list, the simplest method is to enclose it in [. and .] to make it a collating element (see below). Alternatively, make it the first character (following a possible `^`), or (AREs only) precede it with `\`. Alternatively, for `−`, make it the last character, or the second endpoint of a range. To use a literal − as the first endpoint of a range, make it a collating element or (AREs only) precede it with `\`. With the exception of these, some combinations using ] (see next paragraphs), and escapes, all other special characters lose their special significance within a bracket expression.

Within a bracket expression, a collating element (a character, a multi-character sequence that collates as if it were a single character, or a collating-sequence name for either) enclosed in [. and .] stands for the sequence of characters of that collating element. The sequence is a single element of the bracket expression's list. A bracket expression in a locale that has multi-character collating elements can thus match more than one character. So (insidiously), a bracket expression that starts with `^` can match multi-character collating elements even if none of them appear in the bracket expression! (Note: Tcl currently has no multi-character collating elements. This information is only for illustration.)

For example, assume the collating sequence includes a ch multi-character collating element. Then the RE `[[.ch.]]*c` (zero or more ch’s followed by c) matches the first five characters of `chchcc`. Also, the RE `[^c]b` matches all of `chb` (because `[^c]` matches the multi-character ch).

Within a bracket expression, a collating element enclosed in [= and =] is an equivalence class, standing for the sequences of characters of all collating elements equivalent to that one, including itself. (If there are no other equivalent collating elements, the treatment is as if the enclosing delimiters were `[. and `.].) For example, if o and ḍ are the members of an equivalence class, then `[[=o=]]`, `[[=ḍ=]]`, and `[o]` are all synonymous. An equivalence class may not be an endpoint of a range. (Note: Tcl currently implements only the Unicode locale. It doesn't define any equivalence classes. The examples above are just illustrations.)

Within a bracket expression, the name of a character class enclosed in [: and :) stands for the list of all characters (not all collating elements!) belonging to that class. Standard character classes are:

- **alpha** A letter.
- **upper** An upper-case letter.
- **lower** A lower-case letter.
- **digit** A decimal digit.
- **xdigit** A hexadecimal digit.
- **alnum** An alphanumeric (letter or digit).
- **print** An alphanumeric (same as alnum).
- **blank** A space or tab character.
- **space** A character producing white space in displayed text.
- **punct** A punctuation character.
- **graph** A character with a visible representation.
- **cntrl** A control character.
A locale may provide others. (Note that the current Tcl implementation has only one locale: the Unicode locale.) A character class may not be used as an endpoint of a range.

There are two special cases of bracket expressions: the bracket expressions \[[:<:]] and \[[:>:]] are constraints, matching empty strings at the beginning and end of a word respectively. A word is defined as a sequence of word characters that is neither preceded nor followed by word characters. A word character is an *alnum* character or an underscore (_). These special bracket expressions are deprecated; users of AREs should use constraint escapes instead (see below).

### ESCAPES

Escapes (AREs only), which begin with a \ followed by an alphanumeric character, come in several varieties: character entry, class shorthands, constraint escapes, and back references. A \ followed by an alphanumeric character but not constituting a valid escape is illegal in AREs. In EREs, there are no escapes: outside a bracket expression, a \ followed by an alphanumeric character merely stands for that character as an ordinary character, and inside a bracket expression, \ is an ordinary character. (The latter is the one actual incompatibility between EREs and AREs.)

Character–entry escapes (AREs only) exist to make it easier to specify non–printing and otherwise inconvenient characters in REs:

- \a alert (bell) character, as in C
- \b backspace, as in C
- \B synonym for \ to help reduce backslash doubling in some applications where there are multiple levels of backslash processing
- \cX (where X is any character) the character whose low–order 5 bits are the same as those of X, and whose other bits are all zero
- \e the character whose collating–sequence name is `ESC`, or failing that, the character with octal value 033
- \f formfeed, as in C
- \n newline, as in C
- \r carriage return, as in C
- \t horizontal tab, as in C
- \uwxyz (where wxyz is exactly four hexadecimal digits) the Unicode character U+ \uwxyz in the local byte ordering
\Ustuvwxyz  
(where stuvwxyz is exactly eight hexadecimal digits) reserved for a somewhat–hypothetical Unicode extension to 32 bits

\v  
vertical tab, as in C are all available.

\xhhh  
(where hhhh is any sequence of hexadecimal digits) the character whose hexadecimal value is 0xhhh (a single character no matter how many hexadecimal digits are used).

\0  
the character whose value is 0

\xy  
(where xy is exactly two octal digits, and is not a back reference (see below)) the character whose octal value is 0xy

\xyz  
(where xyz is exactly three octal digits, and is not a back reference (see below)) the character whose octal value is 0xyz

Hexadecimal digits are `0`−`9`, `a`−`f`, and `A`−`F`. Octal digits are `0`−`7`.

The character–entry escapes are always taken as ordinary characters. For example, \135 is ] in ASCII, but \135 does not terminate a bracket expression. Beware, however, that some applications (e.g., C compilers) interpret such sequences themselves before the regular–expression package gets to see them, which may require doubling (quadrupling, etc.) the `\`.

Class–shorthand escapes (AREs only) provide shorthands for certain commonly–used character classes:

\d  
[[:digit:]]

\s  
[[:space:]]

\w  
[[:alnum:]_] (note underscore)

\D  
[^[:digit:]]

\S  
[^[:space:]]

\W  
[^[:alnum:]_] (note underscore)

Within bracket expressions, `\d`, `\s`, and `\w` lose their outer brackets, and `\D`, `\S`, and `\W` are illegal. (So, for example, [a–c\d] is equivalent to [a–c[:digit:]]. Also, [a–c\D], which is equivalent to [a–c[^[:digit:]]], is illegal.)

A constraint escape (AREs only) is a constraint, matching the empty string if specific conditions are met, written as an escape:
\A matches only at the beginning of the string (see MATCHING, below, for how this differs from `^')
\m matches only at the beginning of a word
\M matches only at the end of a word
\y matches only at the beginning or end of a word
\Y matches only at a point that is not the beginning or end of a word
\Z matches only at the end of the string (see MATCHING, below, for how this differs from `$')
\m (where \(m\) is a nonzero digit) a back reference, see below
\mn (where \(m\) is a nonzero digit, and \(nn\) is some more digits, and the decimal value \(mnn\) is not greater than the number of closing capturing parentheses seen so far) a back reference, see below

A word is defined as in the specification of [[[:<:]]] and [[[:>:]]] above. Constraint escapes are illegal within bracket expressions.

A back reference (AREs only) matches the same string matched by the parenthesized subexpression specified by the number, so that (e.g.) \(([bc])\1\) matches bb or cc but not `bc'. The subexpression must entirely precede the back reference in the RE. Subexpressions are numbered in the order of their leading parentheses. Non–capturing parentheses do not define subexpressions.

There is an inherent historical ambiguity between octal character–entry escapes and back references, which is resolved by heuristics, as hinted at above. A leading zero always indicates an octal escape. A single non–zero digit, not followed by another digit, is always taken as a back reference. A multi–digit sequence not starting with a zero is taken as a back reference if it comes after a suitable subexpression (i.e. the number is in the legal range for a back reference), and otherwise is taken as octal.

**METASYNTAX**

In addition to the main syntax described above, there are some special forms and miscellaneous syntactic facilities available.

Normally the flavor of RE being used is specified by application–dependent means. However, this can be overridden by a director. If an RE of any flavor begins with `***:', the rest of the RE is an ARE. If an RE of any flavor begins with `***=', the rest of the RE is taken to be a literal string, with all characters considered ordinary characters.

An ARE may begin with embedded options: a sequence (?xyz) (where xyz is one or more alphabetic characters) specifies options affecting the rest of the RE. These supplement, and can override, any options specified by the application. The available option letters are:
rest of RE is a BRE

case-sensitive matching (usual default)

rest of RE is an ERE

case-insensitive matching (see MATCHING, below)

historical synonym for newline-sensitive matching (see MATCHING, below)

partial newline-sensitive matching (see MATCHING, below)

rest of RE is a literal ("quoted") string, all ordinary characters

non-newline-sensitive matching (usual default)

tight syntax (usual default; see below)

inverse partial newline-sensitive ("weird") matching (see MATCHING, below)

expanded syntax (see below)

Embedded options take effect at the ) terminating the sequence. They are available only at the start of an ARE, and may not be used later within it.

In addition to the usual (tight) RE syntax, in which all characters are significant, there is an expanded syntax, available in all flavors of RE with the -expanded switch, or in AREs with the embedded x option. In the expanded syntax, white-space characters are ignored and all characters between a # and the following newline (or the end of the RE) are ignored, permitting paragraphing and commenting a complex RE. There are three exceptions to that basic rule:

a white-space character or `#' preceded by `\' is retained

white space or `#' within a bracket expression is retained

white space and comments are illegal within multi-character symbols like the ARE `(?:' or the BRE `\'

Expanded-syntax white-space characters are blank, tab, newline, and any character that belongs to the space character class.

Finally, in an ARE, outside bracket expressions, the sequence `(?#ttt)' (where ttt is any text not containing a
'\(\text{')\} is a comment, completely ignored. Again, this is not allowed between the characters of multi-character symbols like '\(\text{(??:}\). Such comments are more a historical artifact than a useful facility, and their use is deprecated; use the expanded syntax instead.

None of these metasyntax extensions is available if the application (or an initial \(*\text{**}=\text{ director) has specified that the user's input be treated as a literal string rather than as an RE.

### Matching

In the event that an RE could match more than one substring of a given string, the RE matches the one starting earliest in the string. If the RE could match more than one substring starting at that point, its choice is determined by its **preference**: either the longest substring, or the shortest.

Most atoms, and all constraints, have no preference. A parenthesized RE has the same preference (possibly none) as the RE. A quantified atom with quantifier \(\{m\}\) or \(\{m\}?\) has the same preference (possibly none) as the atom itself. A quantified atom with other normal quantifiers (including \(\{m,n\}\) with \(m\) equal to \(n\)) prefers longest match. A quantified atom with other non-greedy quantifiers (including \(\{m,n\}?\) with \(m\) equal to \(n\)) prefers shortest match. A branch has the same preference as the first quantified atom in it which has a preference. An RE consisting of two or more branches connected by the \(|\) operator prefers longest match.

Subject to the constraints imposed by the rules for matching the whole RE, subexpressions also match the longest or shortest possible substrings, based on their preferences, with subexpressions starting earlier in the RE taking priority over ones starting later. Note that outer subexpressions thus take priority over their component subexpressions.

Note that the quantifiers \(\{1,1\}\) and \(\{1,1\}?\) can be used to force longest and shortest preference, respectively, on a subexpression or a whole RE.

Match lengths are measured in characters, not collating elements. An empty string is considered longer than no match at all. For example, \(bb^*\) matches the three middle characters of `abbbe`, \((\text{week|wee})(\text{night|knights})\) matches all ten characters of `weeknights`, when \((.\text{.})^*\) is matched against `abc` the parenthesized subexpression matches all three characters, and when \((\text{a*})^*\) is matched against `bc` both the whole RE and the parenthesized subexpression match an empty string.

If case-independent matching is specified, the effect is much as if all case distinctions had vanished from the alphabet. When an alphabetic that exists in multiple cases appears as an ordinary character outside a bracket expression, it is effectively transformed into a bracket expression containing both cases, so that x becomes `[xX]`. When it appears inside a bracket expression, all case counterparts of it are added to the bracket expression, so that `[x]` becomes `[xX]` and `[^x]` becomes `[^xX]`.

If newline-sensitive matching is specified, . and bracket expressions using `^` will never match the newline character (so that matches will never cross newlines unless the RE explicitly arranges it) and `^` and `$` will match the empty string after and before a newline respectively, in addition to matching at beginning and end of string respectively. ARE \(\text{\textbackslash A}\) and \(\text{\textbackslash V}\) continue to match beginning or end of string only.
If partial newline–sensitive matching is specified, this affects . and bracket expressions as with newline–sensitive matching, but not ^ and `$.

If inverse partial newline–sensitive matching is specified, this affects ^ and $ as with newline–sensitive matching, but not . and bracket expressions. This isn't very useful but is provided for symmetry.

**LIMITS AND COMPATIBILITY**

No particular limit is imposed on the length of REs. Programs intended to be highly portable should not employ REs longer than 256 bytes, as a POSIX–compliant implementation can refuse to accept such REs.

The only feature of AREs that is actually incompatible with POSIX EREs is that \ does not lose its special significance inside bracket expressions. All other ARE features use syntax which is illegal or has undefined or unspecified effects in POSIX EREs; the *** syntax of directors likewise is outside the POSIX syntax for both BREs and EREs.

Many of the ARE extensions are borrowed from Perl, but some have been changed to clean them up, and a few Perl extensions are not present. Incompatibilities of note include `\b', `\B', the lack of special treatment for a trailing newline, the addition of complemented bracket expressions to the things affected by newline–sensitive matching, the restrictions on parentheses and back references in lookahead constraints, and the longest/shortest–match (rather than first–match) matching semantics.

The matching rules for REs containing both normal and non–greedy quantifiers have changed since early beta–test versions of this package. (The new rules are much simpler and cleaner, but don't work as hard at guessing the user's real intentions.)

Henry Spencer's original 1986 *regexp* package, still in widespread use (e.g., in pre–8.1 releases of Tcl), implemented an early version of today's EREs. There are four incompatibilities between *regexp*'s near–EREs ("RREs" for short) and AREs. In roughly increasing order of significance:

- In AREs, \ followed by an alphanumeric character is either an escape or an error, while in RREs, it was just another way of writing the alphanumeric. This should not be a problem because there was no reason to write such a sequence in RREs.

- { followed by a digit in an ARE is the beginning of a bound, while in RREs, { was always an ordinary character. Such sequences should be rare, and will often result in an error because following characters will not look like a valid bound.

- In AREs, \ remains a special character within `[ ]', so a literal `\ within [ ] must be written `\W. \ also gives a literal `\ within [ ] in RREs, but only truly paranoid programmers routinely doubled the backslash.

- AREs report the longest/shortest match for the RE, rather than the first found in a specified search order. This may affect some RREs which were written in the expectation that the first match would be reported. (The careful crafting of RREs to optimize the search order for fast matching is obsolete.
AREs examine all possible matches in parallel, and their performance is largely insensitive to their complexity but cases where the search order was exploited to deliberately find a match which was not the longest/shortest will need rewriting.

BASIC REGULAR EXPRESSIONS

BREs differ from EREs in several respects. `|`, `+`, and `?` are ordinary characters and there is no equivalent for their functionality. The delimiters for bounds are `{ and `}`, with { and } by themselves ordinary characters. The parentheses for nested subexpressions are `( and `)`, with ( and ) by themselves ordinary characters. `^` is an ordinary character except at the beginning of the RE or the beginning of a parenthesized subexpression, `$` is an ordinary character except at the end of the RE or the end of a parenthesized subexpression, and `*` is an ordinary character if it appears at the beginning of the RE or the beginning of a parenthesized subexpression (after a possible leading `^`). Finally, single-digit back references are available, and `< and >` are synonyms for `[:<:]` and `[:>::]` respectively; no other escapes are available.

SEE ALSO

RegExp, regexp, regsub, lsearch, switch, text

KEYWORDS

match, regular expression, string

Copyright ' 1998 Sun Microsystems, Inc.
Copyright ' 1999 Scriptics Corporation
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

read – Read from a channel

SYNOPSIS

read ?–nonewline? channelId
read channelId numChars

DESCRIPTION

USE WITH SERIAL PORTS

read channelId numChars
read channelId

EXAMPLE

SEE ALSO

KEYWORDS

NAME

read – Read from a channel

SYNOPSIS

read ?–nonewline? channelId
read channelId numChars

DESCRIPTION

In the first form, the read command reads all of the data from channelId up to the end of the file. If the –nonewline switch is specified then the last character of the file is discarded if it is a newline. In the second form, the extra argument specifies how many characters to read. Exactly that many characters will be read and returned, unless there are fewer than numChars left in the file; in this case all the remaining characters are returned. If the channel is configured to use a multi–byte encoding, then the number of characters read may not be the same as the number of bytes read.

ChannelId must be an identifier for an open channel such as the Tcl standard input channel (stdin), the return value from an invocation of open or socket, or the result of a channel creation command provided by a Tcl extension. The channel must have been opened for input.

If channelId is in nonblocking mode, the command may not read as many characters as requested: once all available input has been read, the command will return the data that is available rather than blocking for more input. If the channel is configured to use a multi–byte encoding, then there may actually be some bytes remaining in the internal buffers that do not form a complete character. These bytes will not be returned until a complete character is available or end–of–file is reached. The –nonewline switch is ignored if the command
returns before reaching the end of the file.

**Read** translates end–of–line sequences in the input into newline characters according to the **–translation** option for the channel. See the **fconfigure** manual entry for a discussion on ways in which **fconfigure** will alter input.

**USE WITH SERIAL PORTS**

For most applications a channel connected to a serial port should be configured to be nonblocking: **fconfigure channelId −blocking 0**. Then **read** behaves much like described above. Care must be taken when using **read** on blocking serial ports:

**read** channelId numChars

In this form **read** blocks until **numChars** have been received from the serial port.

**read** channelId

In this form **read** blocks until the reception of the end–of–file character, see **fconfigure −eofchar**. If there no end–of–file character has been configured for the channel, then **read** will block forever.

**EXAMPLE**

This example code reads a file all at once, and splits it into a list, with each line in the file corresponding to an element in the list:

```tcl
set fl [open /proc/meminfo]
set data [read $fl]
close $fl
set lines [split $data \n]
```

**SEE ALSO**

**file, eof, fblocked, fconfigure, Tcl_StandardChannels**

**KEYWORDS**

blocking, channel, end of line, end of file, nonblocking, read, translation, encoding

---

**Copyright** ' 1993 The Regents of the University of California.
**Copyright** ' 1994–1996 Sun Microsystems, Inc.
**Copyright** ' 1995–1997 Roger E. Critchlow Jr.
**NAME**

regexp – Match a regular expression against a string

**SYNOPSIS**

```
regexp ?switches? exp string ?matchVar? ?subMatchVar subMatchVar ...?
```

**DESCRIPTION**

Determines whether the regular expression `exp` matches part or all of `string` and returns 1 if it does, 0 if it doesn't, unless `–inline` is specified (see below). (Regular expression matching is described in the [re_syntax](#) reference page.)

If additional arguments are specified after `string` then they are treated as the names of variables in which to return information about which part(s) of `string` matched `exp`. `MatchVar` will be set to the range of `string` that matched all of `exp`. The first `subMatchVar` will contain the characters in `string` that matched the leftmost parenthesized subexpression within `exp`, the next `subMatchVar` will contain the characters that matched the next parenthesized subexpression to the right in `exp`, and so on.
If the initial arguments to `regexp` start with `−` then they are treated as switches. The following switches are currently supported:

−about

Instead of attempting to match the regular expression, returns a list containing information about the regular expression. The first element of the list is a subexpression count. The second element is a list of property names that describe various attributes of the regular expression. This switch is primarily intended for debugging purposes.

−expanded

Enables use of the expanded regular expression syntax where whitespace and comments are ignored. This is the same as specifying the `(?x)` embedded option (see the `re_syntax` manual page).

−indices

Changes what is stored in the `subMatchVars`. Instead of storing the matching characters from `string`, each variable will contain a list of two decimal strings giving the indices in `string` of the first and last characters in the matching range of characters.

−line

Enables newline-sensitive matching. By default, newline is a completely ordinary character with no special meaning. With this flag, `\[^` bracket expressions and `.` never match newline, `^` matches an empty string after any newline in addition to its normal function, and `$` matches an empty string before any newline in addition to its normal function. This flag is equivalent to specifying both `−linestop` and `−lineanchor`, or the `(？n)` embedded option (see the `re_syntax` manual page).

−linestop

Changes the behavior of `\[^` bracket expressions and `.` so that they stop at newlines. This is the same as specifying the `(？p)` embedded option (see the `re_syntax` manual page).

−lineanchor

Changes the behavior of `^` and `$` (the ``anchors``) so they match the beginning and end of a line respectively. This is the same as specifying the `(？w)` embedded option (see the `re_syntax` manual page).

−nocase

Causes upper–case characters in `string` to be treated as lower case during the matching process.

−all

Causes the regular expression to be matched as many times as possible in the string, returning the total number of matches found. If this is specified with match variables, they will contain information for the last match only.

−inline

Causes the command to return, as a list, the data that would otherwise be placed in match variables. When using `−inline`, match variables may not be specified. If used with `−all`, the list will be concatenated at each iteration, such that a flat list is always returned. For each match iteration, the command will append the overall match data, plus one element for each subexpression in the regular expression. Examples are:

```tcl
regexp −inline −− \{\w(\w)\} " inlined "
=> {in n}
regexp −all −inline −− \{\w(\w)\} " inlined "
=> {in n li i ne e}
```

−start

index
Specifies a character index offset into the string to start matching the regular expression at. When using this switch, `\^` will not match the beginning of the line, and `\A` will still match the start of the string at `index`. If `--indices` is specified, the indices will be indexed starting from the absolute beginning of the input string. `index` will be constrained to the bounds of the input string.

---

Marks the end of switches. The argument following this one will be treated as `exp` even if it starts with a `−`.

If there are more `subMatchVar`'s than parenthesized subexpressions within `exp`, or if a particular subexpression in `exp` doesn't match the string (e.g. because it was in a portion of the expression that wasn't matched), then the corresponding `subMatchVar` will be set to `−1 −1` if `--indices` has been specified or to an empty string otherwise.

**EXAMPLES**

Find the first occurrence of a word starting with `foo` in a string that is not actually an instance of `foobar`, and get the letters following it up to the end of the word into a variable:

```
regexp {\<foo(?!bar\>)(\w*)} $string -> restOfWord
```

Note that the whole matched substring has been placed in the variable `->` which is a name chosen to look nice given that we are not actually interested in its contents.

Find the index of the word `badger` (in any case) within a string and store that in the variable `location`:

```
regexp --indices {(?!i)\<badger\>} $string location
```

Count the number of octal digits in a string:

```
regexp -all {[0-7]} $string
```

List all words (consisting of all sequences of non–whitespace characters) in a string:

```
regexp -all -inline {\S+} $string
```

**SEE ALSO**

`re syntax`, `regsub`

**KEYWORDS**

match, regular expression, string

Copyright 1998 Sun Microsystems, Inc.
Copyright 1995-1997 Roger E. Critchlow Jr.
NAME

registry – Manipulate the Windows registry

SYNOPSIS

package require registry 1.1
registry option keyName ?arg arg ...?

DESCRIPTION

registry broadcast keyName ?−timeout milliseconds?
registry delete keyName ?valueName?
registry get keyName valueName
registry keys keyName ?pattern?
registry set keyName ?valueName data ?type??
registry type keyName valueName
registry values keyName ?pattern?

SUPPORTED TYPES

binary
none
sz
expand_sz
dword
dword_big_endian
link
multi_sz
resource_list

PORTABILITY ISSUES
EXAMPLE
KEYWORDS
Warning: this command should be used with caution as a corrupted registry can leave your system in an unusable state.

**KeyName** is the name of a registry key. Registry keys must be one of the following forms:

\hostname\rootname\keypath

rootname\keypath

rootname

**Hostname** specifies the name of any valid Windows host that exports its registry. The **rootname** component must be one of HKEY_LOCAL_MACHINE, HKEY_USERS, HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_CURRENT_CONFIG, HKEY_PERFORMANCE_DATA, or HKEY_DYN_DATA. The **keypath** can be one or more registry key names separated by backslash (\) characters.

**Option** indicates what to do with the registry key name. Any unique abbreviation for **option** is acceptable. The valid options are:

**registry broadcast** **KeyName** ?–timeout milliseconds?

Sends a broadcast message to the system and running programs to notify them of certain updates. This is necessary to propagate changes to key registry keys like Environment. The timeout specifies the amount of time, in milliseconds, to wait for applications to respond to the broadcast message. It defaults to 3000. The following example demonstrates how to add a path to the global Environment and notify applications of the change without requiring a logoff/logon step (assumes admin privileges):

```tcl
set regPath {HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Environment}
set curPath [registry get $regPath "Path"]
registry set $regPath "Path" "$curPath;$addPath"
registry broadcast "Environment"
```

**registry delete** **KeyName** ?valueName?

If the optional **valueName** argument is present, the specified value under **KeyName** will be deleted from the registry. If the optional **valueName** is omitted, the specified key and any subkeys or values beneath it in the registry hierarchy will be deleted. If the key could not be deleted then an error is generated. If the key did not exist, the command has no effect.

**registry get** **KeyName** **valueName**

Returns the data associated with the value **valueName** under the key **KeyName**. If either the key or the value does not exist, then an error is generated. For more details on the format of the returned data, see SUPPORTED TYPES, below.

**registry keys** **KeyName** ?pattern?

If **pattern** isn't specified, returns a list of names of all the subkeys of **KeyName**. If **pattern** is specified, only those names matching **pattern** are returned. Matching is determined using the same rules as for **string match**. If the specified **KeyName** does not exist, then an error is generated.

**registry set** **KeyName** ?valueName** data **type**?


If `valueName` isn't specified, creates the key `keyName` if it doesn't already exist. If `valueName` is specified, creates the key `keyName` and value `valueName` if necessary. The contents of `valueName` are set to `data` with the type indicated by `type`. If `type` isn't specified, the type `sz` is assumed. For more details on the data and type arguments, see SUPPORTED TYPES below.

`registry type keyName valueName`  
Returns the type of the value `valueName` in the key `keyName`. For more information on the possible types, see SUPPORTED TYPES, below.

`registry values keyName ?pattern?`  
If `pattern` isn't specified, returns a list of names of all the values of `keyName`. If `pattern` is specified, only those names matching `pattern` are returned. Matching is determined using the same rules as for `string match`.

**SUPPORTED TYPES**

Each value under a key in the registry contains some data of a particular type in a type–specific representation. The `registry` command converts between this internal representation and one that can be manipulated by Tcl scripts. In most cases, the data is simply returned as a Tcl string. The type indicates the intended use for the data, but does not actually change the representation. For some types, the `registry` command returns the data in a different form to make it easier to manipulate. The following types are recognized by the registry command:

- **binary**
  The registry value contains arbitrary binary data. The data is represented exactly in Tcl, including any embedded nulls.

- **none**
  The registry value contains arbitrary binary data with no defined type. The data is represented exactly in Tcl, including any embedded nulls.

- **sz**
  The registry value contains a null–terminated string. The data is represented in Tcl as a string.

- **expand_sz**
  The registry value contains a null–terminated string that contains unexpanded references to environment variables in the normal Windows style (for example, "%PATH%"). The data is represented in Tcl as a string.

- **dword**
  The registry value contains a little–endian 32–bit number. The data is represented in Tcl as a decimal string.

- **dword_big_endian**
  The registry value contains a big–endian 32–bit number. The data is represented in Tcl as a decimal string.

- **link**
  The registry value contains a symbolic link. The data is represented exactly in Tcl, including any embedded nulls.

- **multi_sz**
  The registry value contains an array of null–terminated strings. The data is represented in Tcl as a list of strings.
resource_list

The registry value contains a device–driver resource list. The data is represented exactly in Tcl, including any embedded nulls.

In addition to the symbolically named types listed above, unknown types are identified using a 32–bit integer that corresponds to the type code returned by the system interfaces. In this case, the data is represented exactly in Tcl, including any embedded nulls.

PORTABILITY ISSUES

The registry command is only available on Windows.

EXAMPLE

Print out how double–clicking on a Tcl script file will invoke a Tcl interpreter:

```tcl
package require registry
set ext .tcl

# Read the type name
set type [registry get HKEY_CLASSES_ROOT\$ext {}]
# Work out where to look for the command
set path HKEY_CLASSES_ROOT\$type\Shell\Open\command
# Read the command!
set command [registry get $path {}]

puts "$ext opens with $command"
```

KEYWORDS

registry

Copyright ' 1997 Sun Microsystems, Inc.
Copyright ' 2002 ActiveState Corporation.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

regsub – Perform substitutions based on regular expression pattern matching

SYNOPSIS

regsub ?switches? exp string subSpec ?varName?

DESCRIPTION

This command matches the regular expression exp against string, and either copies string to the variable whose name is given by varName or returns string if varName is not present. (Regular expression matching is described in the re_syntax reference page.) If there is a match, then while copying string to varName (or to the result of this command if varName is not present) the portion of string that matched exp is replaced with subSpec. If subSpec contains a `&` or `\0`, then it is replaced in the substitution with the portion of string that matched exp. If subSpec contains a `\n`, where n is a digit between 1 and 9, then it is replaced in the substitution with the portion of string that matched the n-th parenthesized subexpression of exp. Additional backslashes may be used in subSpec to prevent special interpretation of `&` or `\0` or `\n` or backslash. The use of backslashes in subSpec tends to interact badly with the Tcl parser's use of backslashes, so it's generally safest to enclose subSpec in braces if it includes backslashes.

If the initial arguments to regsub start with – then they are treated as switches. The following switches are currently supported:
All ranges in string that match exp are found and substitution is performed for each of these ranges. Without this switch only the first matching range is found and substituted. If –all is specified, then `"&" and `"\n" sequences are handled for each substitution using the information from the corresponding match.

–expanded

Enables use of the expanded regular expression syntax where whitespace and comments are ignored. This is the same as specifying the (?x) embedded option (see the re_syntax manual page).

–line

Enables newline–sensitive matching. By default, newline is a completely ordinary character with no special meaning. With this flag, `[\^' bracket expressions and `.' never match newline, `\' matches an empty string after any newline in addition to its normal function, and `$' matches an empty string before any newline in addition to its normal function. This flag is equivalent to specifying both –linestop and –lineanchor, or the (?n) embedded option (see the re_syntax manual page).

–linestop

Changes the behavior of `[^' bracket expressions and `.' so that they stop at newlines. This is the same as specifying the (?p) embedded option (see the re_syntax manual page).

–lineanchor

Changes the behavior of `^' and `$' (the `anchors") so they match the beginning and end of a line respectively. This is the same as specifying the (?w) embedded option (see the re_syntax manual page).

–nocase

Upper–case characters in string will be converted to lower–case before matching against exp; however, substitutions specified by subSpec use the original unconverted form of string.

–start index

Specifies a character index offset into the string to start matching the regular expression at. When using this switch, `[^' will not match the beginning of the line, and \A will still match the start of the string at index. index will be constrained to the bounds of the input string.

–

Marks the end of switches. The argument following this one will be treated as exp even if it starts with a –.

If varName is supplied, the command returns a count of the number of matching ranges that were found and replaced, otherwise the string after replacement is returned. See the manual entry for regexp for details on the interpretation of regular expressions.

EXAMPLES

Replace (in the string in variable string) every instance of foo which is a word by itself with bar:

regsub –all {<foo>} $string bar string

Insert double–quotes around the first instance of the word interesting, however it is capitalised.

regsub –nocase {<interesting>} $string{"&"} string
Convert all non-ASCII and Tcl-significant characters into \u escape sequences by using `regsub` and `subst` in combination:

```tcl
# This RE is just a character class for everything "bad"
set RE {[]{}\\$\u0100−\uffff}

# We will substitute with a fragment of Tcl script in brackets
set substitution {\{format \\\\u%04x [scan "\\&" %c]\}}

# Now we apply the substitution to get a subst-string that
# will perform the computational parts of the conversion.
set quoted [subst [regsub -all $RE $string $substitution]]
```

SEE ALSO

`regexp`, `re_syntax`, `subst`

KEYWORDS

match, pattern, regular expression, substitute

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 2000 Scriptics Corporation.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
rename

NAME

rename – Rename or delete a command

SYNOPSIS

rename oldName newName

DESCRIPTION

Rename the command that used to be called *oldName* so that it is now called *newName*. If *newName* is an empty string then *oldName* is deleted. *oldName* and *newName* may include namespace qualifiers (names of containing namespaces). If a command is renamed into a different namespace, future invocations of it will execute in the new namespace. The rename command returns an empty string as result.

EXAMPLE

The rename command can be used to wrap the standard Tcl commands with your own monitoring machinery. For example, you might wish to count how often the source command is called:

```tcl
rename ::source ::theRealSource
set sourceCount 0
proc ::source args {
    global sourceCount
    puts "called source for the [incr sourceCount]'th time"
    uplevel 1 ::theRealSource $args
}
```

SEE ALSO

namespace, proc

KEYWORDS

command, delete, namespace, rename

---

Copyright © 1993 The Regents of the University of California.  
Copyright © 1994-1997 Sun Microsystems, Inc.  
Copyright © 1995-1997 Roger E. Critchlow Jr.
resource

NAME

resource – Manipulate Macintosh resources

SYNOPSIS

resource option ?arg arg ...?

DESCRIPTION

resource close rsrcRef
resource delete ?options? resourceType
   -id resourceId
   -name resourceName
   -file resourceRef
resource files ?resourceRef?
resource list resourceType ?resourceRef?
resource open fileName ?access?
resource read resourceType resourceId ?resourceRef?
resource types ?resourceRef?
resource write ?options? resourceType data
   -id resourceId
   -name resourceName
   -file resourceRef
   -force

RESOURCE TYPES
RESOURCE IDS
PORTABILITY ISSUES
SEE ALSO
KEYWORDS

NAME

resource – Manipulate Macintosh resources

SYNOPSIS

resource option ?arg arg ...?

DESCRIPTION

The resource command provides some generic operations for dealing with Macintosh resources. This command is only supported on the Macintosh platform. Each Macintosh file consists of two forks: a data fork and a resource fork. You use the normal open, puts, close, etc. commands to manipulate the data fork. You must use this command, however, to interact with the resource fork. Option indicates what resource command
to perform. Any unique abbreviation for option is acceptable. The valid options are:

**resource close rsrcRef**
Closes the given resource reference (obtained from resource open). Resources from that resource file will no longer be available.

**resource delete ?options? resourceType**
This command will delete the resource specified by options and type resourceType (see RESOURCE TYPES below). The options give you several ways to specify the resource to be deleted.

---

**–id resourceId**
If the –id option is given the id resourceId (see RESOURCE IDS below) is used to specify the resource to be deleted. The id must be a number – to specify a name use the –name option.

**–name resourceName**
If –name is specified, the resource named resourceName will be deleted. If the –id is also provided, then there must be a resource with BOTH this name and this id. If no name is provided, then the id will be used regardless of the name of the actual resource.

**–file resourceRef**
If the –file option is specified then the resource will be deleted from the file pointed to by resourceRef. Otherwise the first resource with the given resourceName and or resourceId which is found on the resource file path will be deleted. To inspect the file path, use the resource files command.

**resource files ?resourceRef?**
If resourceRef is not provided, this command returns a Tcl list of the resource references for all the currently open resource files. The list is in the normal Macintosh search order for resources. If resourceRef is specified, the command will return the path to the file whose resource fork is represented by that token.

**resource list resourceType ?resourceRef?**
List all of the resources ids of type resourceType (see RESOURCE TYPES below). If resourceRef is specified then the command will limit the search to that particular resource file. Otherwise, all resource files currently opened by the application will be searched. A Tcl list of either the resource name's or resource id's of the found resources will be returned. See the RESOURCE IDS section below for more details about what a resource id is.

**resource open fileName ?access?**
Open the resource for the file fileName. Standard file access permissions may also be specified (see the manual entry for open for details). A resource reference (resourceRef) is returned that can be used by the other resource commands. An error can occur if the file doesn't exist or the file does not have a resource fork. However, if you open the file with write permissions the file and/or resource fork will be created instead of generating an error.

**resource read resourceType resourceId ?resourceRef?**
Read the entire resource of type resourceType (see RESOURCE TYPES below) and the name or id of resourceId (see RESOURCE IDS below) into memory and return the result. If resourceRef is specified we limit our search to that resource file, otherwise we search all open resource forks in the application. It is important to note that most Macintosh resource use a binary format and the data returned from this command may have embedded NULLs or other non-ASCII data.
resource types ?resourceRef?

This command returns a Tcl list of all resource types (see RESOURCE TYPES below) found in the resource file pointed to by resourceRef. If resourceRef is not specified it will return all the resource types found in every resource file currently opened by the application.

resource write ?options? resourceType data

This command will write the passed in data as a new resource of type resourceType (see RESOURCE TYPES below). Several options are available that describe where and how the resource is stored.

−id resourceId
If the −id option is given the id resourceId (see RESOURCE IDS below) is used for the new resource, otherwise a unique id will be generated that will not conflict with any existing resource. However, the id must be a number – to specify a name use the −name option.

−name resourceName
If −name is specified the resource will be named resourceName, otherwise it will have the empty string as the name.

−file resourceRef
If the −file option is specified then the resource will be written in the file pointed to by resourceRef, otherwise the most recently open resource will be used.

−force
If the target resource already exists, then by default Tcl will not overwrite it, but raise an error instead. Use the −force flag to force overwriting the extant resource.

RESOURCE TYPES

Resource types are defined as a four character string that is then mapped to an underlying id. For example, TEXT refers to the Macintosh resource type for text. The type STR# is a list of counted strings. All Macintosh resources must be of some type. See Macintosh documentation for a more complete list of resource types that are commonly used.

RESOURCE IDS

For this command the notion of a resource id actually refers to two ideas in Macintosh resources. Every place you can use a resource Id you can use either the resource name or a resource number. Names are always searched or returned in preference to numbers. For example, the resource list command will return names if they exist or numbers if the name is NULL.

PORTABILITY ISSUES

The resource command is only available on Macintosh.

SEE ALSO

open
KEYWORDS

open, resource

Copyright © 1997 Sun Microsystems, Inc.
Copyright © 1995-1997 Roger E. Critchlow Jr.
NAME

return − Return from a procedure

SYNOPSIS


DESCRIPTION

EXCEPTIONAL RETURN CODES

ok (or 0)
error (1)
return (2)
break (3)
continue (4)
value

EXAMPLES

SEE ALSO

KEYWORDS
Error return: the return code of the procedure is 1 (TCL_ERROR). The procedure command behaves in its calling context as if it were the command error result. See below for additional options.

return (2)
The return code of the procedure is 2 (TCL_RETURN). The procedure command behaves in its calling context as if it were the command return (with no arguments).

break (3)
The return code of the procedure is 3 (TCL_BREAK). The procedure command behaves in its calling context as if it were the command break.

continue (4)
The return code of the procedure is 4 (TCL_CONTINUE). The procedure command behaves in its calling context as if it were the command continue.

value
Value must be an integer; it will be returned as the return code for the current procedure.

The −code option is rarely used. It is provided so that procedures that implement new control structures can reflect exceptional conditions back to their callers.

Two additional options, −errorinfo and −errorcode, may be used to provide additional information during error returns. These options are ignored unless code is error.

The −errorinfo option specifies an initial stack trace for the errorInfo variable; if it is not specified then the stack trace left in errorInfo will include the call to the procedure and higher levels on the stack but it will not include any information about the context of the error within the procedure. Typically the info value is supplied from the value left in errorInfo after a catch command trapped an error within the procedure.

If the −errorcode option is specified then code provides a value for the errorCode variable. If the option is not specified then errorCode will default to NONE.

EXAMPLES

First, a simple example of using return to return from a procedure, interrupting the procedure body.

```tcl
proc printOneLine {} {
    puts "line 1" ;# This line will be printed.
    return
    puts "line 2" ;# This line will not be printed.
}
```

Next, an example of using return to set the value returned by the procedure.

```tcl
proc returnX {} {return X}
puts [returnX] ;# prints "X"
```

Next, a more complete example, using return −code error to report invalid arguments.

```tcl
proc factorial {n} {
    if {![string is integer $n] || ($n < 0)} {
```

EXAMPLES
return -code error \
  "expected non-negative integer,\n  but got "$n""
}
if {$n < 2} {
  return 1
}
set m [expr {$n - 1}]
set code [catch {factorial $m} factor]
if {$code != 0} {
  return -code $code $factor
}
set product [expr {$n * $factor}]
if {$product < 0} {
  return -code error \
    "overflow computing factorial of $n"
}
return $product

Next, a procedure replacement for break.

proc myBreak {} {
  return -code break
}

SEE ALSO

break, catch, continue, error, proc, source, telvars

KEYWORDS

break, catch, continue, error, procedure, return

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

scan – Parse string using conversion specifiers in the style of scanf

SYNOPSIS

scan string format ?varName varName ...?

INTRODUCTION

Details on scanning

d

do

ux

i

c

se
en or f or g
[chars]
[^chars]

DIFFERENCES FROM ANSI SSCANF

EXAMPLES

SEE ALSO

KEYWORDS

NAME

scan – Parse string using conversion specifiers in the style of scanf

SYNOPSIS

scan string format ?varName varName ...?

INTRODUCTION

This command parses fields from an input string in the same fashion as the ANSI C sscanf procedure and returns a count of the number of conversions performed, or −1 if the end of the input string is reached before any conversions have been performed. String gives the input to be parsed and format indicates how to parse it, using % conversion specifiers as in sscanf. Each varName gives the name of a variable; when a field is scanned from string the result is converted back into a string and assigned to the corresponding variable. If no varName variables are specified, then scan works in an inline manner, returning the data that would otherwise be stored in the variables as a list. In the inline case, an empty string is returned when the end of the input string is reached before any conversions have been performed.
DETAILS ON SCANNING

Scan operates by scanning string and format together. If the next character in format is a blank or tab then it matches any number of white space characters in string (including zero). Otherwise, if it isn’t a % character then it must match the next character of string. When a % is encountered in format, it indicates the start of a conversion specifier. A conversion specifier contains up to four fields after the %: a *, which indicates that the converted value is to be discarded instead of assigned to a variable; a XPG3 position specifier; a number indicating a maximum field width; a field size modifier; and a conversion character. All of these fields are optional except for the conversion character. The fields that are present must appear in the order given above.

When scan finds a conversion specifier in format, it first skips any white−space characters in string (unless the specifier is [ or ]). Then it converts the next input characters according to the conversion specifier and stores the result in the variable given by the next argument to scan.

If the % is followed by a decimal number and a $, as in ``%2$d'', then the variable to use is not taken from the next sequential argument. Instead, it is taken from the argument indicated by the number, where 1 corresponds to the first varName. If there are any positional specifiers in format then all of the specifiers must be positional. Every varName on the argument list must correspond to exactly one conversion specifier or an error is generated, or in the inline case, any position can be specified at most once and the empty positions will be filled in with empty strings.

The following conversion characters are supported:

\[ d \]

The input field must be a decimal integer. It is read in and the value is stored in the variable as a decimal string. If the I or L field size modifier is given, the scanned value will have an internal representation that is at least 64−bits in size.

\[ o \]

The input field must be an octal integer. It is read in and the value is stored in the variable as a decimal string. If the I or L field size modifier is given, the scanned value will have an internal representation that is at least 64−bits in size. If the value exceeds MAX_INT (017777777777 on platforms using 32−bit integers when the I and L modifiers are not given), it will be truncated to a signed integer. Hence, 037777777777 will appear as −1 on a 32−bit machine by default.

\[ x \]

The input field must be a hexadecimal integer. It is read in and the value is stored in the variable as a decimal string. If the I or L field size modifier is given, the scanned value will have an internal representation that is at least 64−bits in size. If the value exceeds MAX_INT (0x7FFFFFFF on platforms using 32−bit integers when the I and L modifiers are not given), it will be truncated to a signed integer. Hence, 0xFFFFFFFF will appear as −1 on a 32−bit machine.

\[ u \]

The input field must be a decimal integer. The value is stored in the variable as an unsigned decimal integer string. If the I or L field size modifier is given, the scanned value will have an internal representation that is at least 64−bits in size.

\[ i \]

The input field must be an integer. The base (i.e. decimal, octal, or hexadecimal) is determined in the
same fashion as described in `expr`. The value is stored in the variable as a decimal string. If the `l` or `L` field size modifier is given, the scanned value will have an internal representation that is at least 64–bits in size.

`c`

A single character is read in and its binary value is stored in the variable as a decimal string. Initial white space is not skipped in this case, so the input field may be a white–space character. This conversion is different from the ANSI standard in that the input field always consists of a single character and no field width may be specified.

`s`

The input field consists of all the characters up to the next white–space character; the characters are copied to the variable.

`e` or `f` or `g`

The input field must be a floating–point number consisting of an optional sign, a string of decimal digits possibly containing a decimal point, and an optional exponent consisting of an `e` or `E` followed by an optional sign and a string of decimal digits. It is read in and stored in the variable as a floating–point string.

`[chars]`

The input field consists of any number of characters in `chars`. The matching string is stored in the variable. If the first character between the brackets is a `]` then it is treated as part of `chars` rather than the closing bracket for the set. If `chars` contains a sequence of the form `a−b` then any character between `a` and `b` (inclusive) will match. If the first or last character between the brackets is a `−`, then it is treated as part of `chars` rather than indicating a range.

`[^chars]`

The input field consists of any number of characters not in `chars`. The matching string is stored in the variable. If the character immediately following the `^` is a `]` then it is treated as part of the set rather than the closing bracket for the set. If `chars` contains a sequence of the form `a−b` then any character between `a` and `b` (inclusive) will be excluded from the set. If the first or last character between the brackets is a `−`, then it is treated as part of `chars` rather than indicating a range.

`n`

No input is consumed from the input string. Instead, the total number of characters scanned from the input string so far is stored in the variable.

The number of characters read from the input for a conversion is the largest number that makes sense for that particular conversion (e.g. as many decimal digits as possible for `%d`, as many octal digits as possible for `%o`, and so on). The input field for a given conversion terminates either when a white–space character is encountered or when the maximum field width has been reached, whichever comes first. If a `*` is present in the conversion specifier then no variable is assigned and the next scan argument is not consumed.

DIFFERENCES FROM ANSI `SSCANF`

The behavior of the `scan` command is the same as the behavior of the ANSI C `sscanf` procedure except for the following differences:

`[1]`

`%p` conversion specifier is not currently supported.
For %c conversions a single character value is converted to a decimal string, which is then assigned to the corresponding varName; no field width may be specified for this conversion.

The h modifier is always ignored and the l and L modifiers are ignored when converting real values (i.e. type double is used for the internal representation).

If the end of the input string is reached before any conversions have been performed and no variables are given, an empty string is returned.

EXAMPLES

Parse a simple color specification of the form #RRGGBB using hexadecimal conversions with field sizes:

```tcl
set string "#08D03F"
scan $string "#%2x%2x%2x" r g b
```

Parse a HH:MM time string, noting that this avoids problems with octal numbers by forcing interpretation as decimals (if we did not care, we would use the %i conversion instead):

```tcl
set string "08:08" ;# *Not* octal!
if {{scan $string "%d:%d" hours minutes} != 2} {
  error "not a valid time string"
} # We have to understand numeric ranges ourselves...
if {($minutes < 0 || $minutes > 59)} {
  error "invalid number of minutes"
}
```

Break a string up into sequences of non–whitespace characters (note the use of the %n conversion so that we get skipping over leading whitespace correct):

```tcl
set string " a string {with braced words} + leading space "
set words ()
while {{scan $string %s%n word length} == 2} {
  lappend words $word
  set string [string range $string $length end]
}
```

Parse a simple coordinate string, checking that it is complete by looking for the terminating character explicitly:

```tcl
set string "(5.2,−4e−2)"
# Note that the spaces before the literal parts of
# the scan pattern are significant, and that ")" is
# the Unicode character \u0029
if {
  [scan $string " (%f,%f %c" x y last] != 3
    || $last != 0x0029
```
} then {
    error "invalid coordinate string"
}
puts "X=$x, Y=$y"

SEE ALSO

format, scanf

KEYWORDS

conversion specifier, parse, scan

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 2000 Scriptics Corporation.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
seek

NAME

seek – Change the access position for an open channel

SYNOPSIS

seek channelId offset ?origin?

DESCRIPTION

Changes the current access position for channelId.

ChannelId must be an identifier for an open channel such as a Tcl standard channel (stdin, stdout, or stderr), the return value from an invocation of open or socket, or the result of a channel creation command provided by a Tcl extension.

The offset and origin arguments specify the position at which the next read or write will occur for channelId. Offset must be an integer (which may be negative) and origin must be one of the following:

start

The new access position will be offset bytes from the start of the underlying file or device.

current

The new access position will be offset bytes from the current access position; a negative offset moves the access position backwards in the underlying file or device.

end

The new access position will be offset bytes from the end of the file or device. A negative offset places the access position before the end of file, and a positive offset places the access position after the end of file.
The argument defaults to start.

The command flushes all buffered output for the channel before the command returns, even if the channel is in nonblocking mode. It also discards any buffered and unread input. This command returns an empty string. An error occurs if this command is applied to channels whose underlying file or device does not support seeking.

Note that offset values are byte offsets, not character offsets. Both seek and tell operate in terms of bytes, not characters, unlike read.

**EXAMPLES**

Read a file twice:

```bash
set f [open file.txt]
set data1 [read $f]
seek $f 0
set data2 [read $f]
close $f
# $data1 == $data2 if the file wasn't updated
```

Read the last 10 bytes from a file:

```bash
set f [open file.data]
# This is guaranteed to work with binary data but
# may fail with other encodings...
fconfigure $f -translation binary
seek $f -10 end
set data [read $f 10]
close $f
```

**SEE ALSO**

file, open, close, gets, tell, Tcl_StandardChannels

**KEYWORDS**

access position, file, seek

---

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
set

NAME

set – Read and write variables

SYNOPSIS

set varName ?value?

DESCRIPTION

Returns the value of variable varName. If value is specified, then set the value of varName to value, creating a new variable if one doesn't already exist, and return its value. If varName contains an open parenthesis and ends with a close parenthesis, then it refers to an array element: the characters before the first open parenthesis are the name of the array, and the characters between the parentheses are the index within the array. Otherwise varName refers to a scalar variable.

If varName includes namespace qualifiers (in the array name if it refers to an array element), or if varName is unqualified (does not include the names of any containing namespaces) but no procedure is active, varName refers to a namespace variable resolved according to the rules described under NAME RESOLUTION in the namespace manual page.

If a procedure is active and varName is unqualified, then varName refers to a parameter or local variable of the procedure, unless varName was declared to resolve differently through one of the global, variable or upvar commands.

EXAMPLES

Store a random number in the variable r:

    set r [expr rand()]  

Store a short message in an array element:

    set anAry(msg) "Hello, World!"  

Store a short message in an array element specified by a variable:

    set elemName "msg"  
    set anAry($elemName) "Hello, World!"

Copy a value into the variable out from a variable whose name is stored in the vbl (note that it is often easier to use arrays in practice instead of doing double–dereferencing):

    set
set in0 "small random"
set in1 "large random"
set vbl in[expr {rand()} >= 0.5]
set out [set $vbl]

SEE ALSO

expr, global, namespace, proc, trace, unset, upvar, variable

KEYWORDS

read, write, variable

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

socket – Open a TCP network connection

SYNOPSIS

socket ?options? host port
socket –server command ?options? port

DESCRIPTION

CLIENT SOCKETS
−myaddr addr
−myport port
−async

SERVER SOCKETS
−myaddr addr

CONFIGURATION OPTIONS
−error
−sockname
−peername

EXAMPLES

SEE ALSO

KEYWORDS

NAME

socket – Open a TCP network connection

SYNOPSIS

socket ?options? host port
socket –server command ?options? port

DESCRIPTION

This command opens a network socket and returns a channel identifier that may be used in future invocations of commands like read, puts and flush. At present only the TCP network protocol is supported; future releases may include support for additional protocols. The socket command may be used to open either the client or server side of a connection, depending on whether the –server switch is specified.

Note that the default encoding for all sockets is the system encoding, as returned by encoding system. Most of the time, you will need to use fconfigure to alter this to something else, such as utf−8 (ideal for communicating with other Tcl processes) or iso8859−1 (useful for many network protocols, especially the older ones).
CLIENT SOCKETS

If the −server option is not specified, then the client side of a connection is opened and the command returns a channel identifier that can be used for both reading and writing. Port and host specify a port to connect to; there must be a server accepting connections on this port. Port is an integer port number (or service name, where supported and understood by the host operating system) and host is either a domain–style name such as www.tcl.tk or a numerical IP address such as 127.0.0.1. Use localhost to refer to the host on which the command is invoked.

The following options may also be present before host to specify additional information about the connection:

−myaddr addr
Addr gives the domain–style name or numerical IP address of the client–side network interface to use for the connection. This option may be useful if the client machine has multiple network interfaces. If the option is omitted then the client–side interface will be chosen by the system software.

−myport port
Port specifies an integer port number (or service name, where supported and understood by the host operating system) to use for the client's side of the connection. If this option is omitted, the client's port number will be chosen at random by the system software.

−async
The −async option will cause the client socket to be connected asynchronously. This means that the socket will be created immediately but may not yet be connected to the server, when the call to socket returns. When a gets or flush is done on the socket before the connection attempt succeeds or fails, if the socket is in blocking mode, the operation will wait until the connection is completed or fails. If the socket is in nonblocking mode and a gets or flush is done on the socket before the connection attempt succeeds or fails, the operation returns immediately and fblocked on the socket returns 1.

SERVER SOCKETS

If the −server option is specified then the new socket will be a server for the port given by port (either an integer or a service name, where supported and understood by the host operating system; if port is zero, the operating system will allocate a free port to the server socket which may be discovered by using fconfigure to read the −sockname option). Tcl will automatically accept connections to the given port. For each connection Tcl will create a new channel that may be used to communicate with the client. Tcl then invokes command with three additional arguments: the name of the new channel, the address, in network address notation, of the client's host, and the client's port number.

The following additional option may also be specified before host:

−myaddr addr
Addr gives the domain–style name or numerical IP address of the server–side network interface to use for the connection. This option may be useful if the server machine has multiple network interfaces. If the option is omitted then the server socket is bound to the special address INADDR_ANY so that it can accept connections from any interface.
Server channels cannot be used for input or output; their sole use is to accept new client connections. The channels created for each incoming client connection are opened for input and output. Closing the server channel shuts down the server so that no new connections will be accepted; however, existing connections will be unaffected.

Server sockets depend on the Tcl event mechanism to find out when new connections are opened. If the application doesn't enter the event loop, for example by invoking the `vwait` command or calling the C procedure `Tcl_DoOneEvent`, then no connections will be accepted.

If `port` is specified as zero, the operating system will allocate an unused port for use as a server socket. The port number actually allocated may be retrieved from the created server socket using the `fconfigure` command to retrieve the `−sockname` option as described below.

### CONFIGURATION OPTIONS

The `fconfigure` command can be used to query several readonly configuration options for socket channels:

−error
This option gets the current error status of the given socket. This is useful when you need to determine if an asynchronous connect operation succeeded. If there was an error, the error message is returned. If there was no error, an empty string is returned.

−sockname
This option returns a list of three elements, the address, the host name and the port number for the socket. If the host name cannot be computed, the second element is identical to the address, the first element of the list.

−peername
This option is not supported by server sockets. For client and accepted sockets, this option returns a list of three elements; these are the address, the host name and the port to which the peer socket is connected or bound. If the host name cannot be computed, the second element of the list is identical to the address, its first element.

### EXAMPLES

Here is a very simple time server:

```tcl
proc Server {channel clientaddr clientport} {
    puts "Connection from $clientaddr registered"
    puts $channel [clock format [clock seconds]]
    close $channel
}

socket -server Server 9900
vwait forever
```

And here is the corresponding client to talk to the server:

```tcl
set server localhost
```
set sockChan [socket $server 9900]
gets $sockChan line
close $sockChan
puts "The time on $server is $line"

SEE ALSO

fconfigure, flush, open, read

KEYWORDS

bind, channel, connection, domain name, host, network address, socket, tcp

Copyright ' 1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
source

NAME

source – Evaluate a file or resource as a Tcl script

SYNOPSIS

source fileName
source –rsrc resourceName ?fileName?
source –rsrid resourceId ?fileName?

DESCRIPTION

This command takes the contents of the specified file or resource and passes it to the Tcl interpreter as a text script. The return value from source is the return value of the last command executed in the script. If an error occurs in evaluating the contents of the script then the source command will return that error. If a return command is invoked from within the script then the remainder of the file will be skipped and the source command will return normally with the result from the return command.

The end–of–file character for files is `\32' (^Z) for all platforms. The source command will read files up to this character. This restriction does not exist for the read or gets commands, allowing for files containing code and data segments (scripted documents). If you require a ``^Z'' in code for string comparison, you can use ``\032'' or ``\u001a'', which will be safely substituted by the Tcl interpreter into ``^Z''.

The –rsrc and –rsrid forms of this command are only available on Macintosh computers. These versions of the command allow you to source a script from a TEXT resource. You may specify what TEXT resource to source by either name or id. By default Tcl searches all open resource files, which include the current application and any loaded C extensions. Alternatively, you may specify the fileName where the TEXT resource can be found.

EXAMPLE

Run the script in the file foo.tcl and then the script in the file bar.tcl:

source foo.tcl
source bar.tcl

Alternatively:

foreach scriptFile {foo.tcl bar.tcl} {
    source $scriptFile
}
SEE ALSO

file, cd, info

KEYWORDS

file, script

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 2000 Scriptics Corporation.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
split

NAME

split – Split a string into a proper Tcl list

SYNOPSIS

split string ?splitChars?

DESCRIPTION

Returns a list created by splitting string at each character that is in the splitChars argument. Each element of the result list will consist of the characters from string that lie between instances of the characters in splitChars. Empty list elements will be generated if string contains adjacent characters in splitChars, or if the first or last character of string is in splitChars. If splitChars is an empty string then each character of string becomes a separate element of the result list. SplitChars defaults to the standard white-space characters.

EXAMPLES

Divide up a USENET group name into its hierarchical components:

split "comp.lang.tcl.announce" .
=> comp lang tcl announce

See how the split command splits on every character in splitChars, which can result in information loss if you are not careful:

split "alpha beta gamma" "temp"
=> al {ha b} {} {a ga} {} a

Extract the list words from a string that is not a well-formed list:

split "Example with \{unbalanced brace character"
=> Example with \\unbalanced brace character

Split a string into its constituent characters

split "Hello world" {}
=> Hello world

PARSING RECORD-ORIENTED FILES

Parse a Unix /etc/passwd file, which consists of one entry per line, with each line consisting of a colon-separated list of fields:
## Read the file
```tcl
set fid [open /etc/passwd]
set content [read $fid]
close $fid
```

## Split into records on newlines
```tcl
set records [split $content "\n"]
```

## Iterate over the records
```tcl
foreach rec $records {
    # Split into fields on colons
    set fields [split $rec ":"]

    # Assign fields to variables and print some out...
    lassign $fields \
            userName password uid grp longName homeDir shell
    puts "$longName uses [file tail $shell] for a login shell"
}
```

### SEE ALSO

`join`, `list`, `string`

### KEYWORDS

`list`, `split`, `string`

---

`Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.`
NAME

string – Manipulate strings

SYNOPSIS

string option arg ?arg ...? 

DESCRIPTION

string bytelength string
string compare ?–nocase? ?–length int? string1 string2
string equal ?–nocase? ?–length int? string1 string2
string first string1 string2 ?startIndex?
string index string charIndex
    integer
    end
    end–integer
string is class ?–strict? ?–failindex varname? string
    alnum
    alpha
    ascii
    boolean
    control
    digit
    double
    false
    graph
    integer
    lower
    print
    punct
    space
    true
    upper
    wordchar
    xdigit

string last string1 string2 ?lastIndex?

string length string

string map ?–nocase? mapping string

string match ?–nocase? pattern string

*  
?  
[chars]
/$

string range string first last
string repeat string count
string replace string first last ?newstring?
string tolower string ?first? ?last?
string totitle string ?first? ?last?
string toupper string ?first? ?last?
string trim string ?chars?
string trimleft string ?chars?
string trimright string ?chars?
string wordend string charIndex
string wordstart string charIndex

EXAMPLE
SEE ALSO
KEYWORDS

NAME

string – Manipulate strings

SYNOPSIS

string option arg ?arg ...?

DESCRIPTION

Performs one of several string operations, depending on option. The legal options (which may be abbreviated) are:

string bytelength string
Returns a decimal string giving the number of bytes used to represent string in memory. Because UTF–8 uses one to three bytes to represent Unicode characters, the byte length will not be the same as the character length in general. The cases where a script cares about the byte length are rare. In almost all cases, you should use the string length operation (including determining the length of a Tcl ByteArray object). Refer to the Tcl_NumUtfChars manual entry for more details on the UTF–8 representation.

string compare ?nocase? ?length int? string1 string2
Perform a character–by–character comparison of strings string1 and string2. Returns –1, 0, or 1, depending on whether string1 is lexicographically less than, equal to, or greater than string2. If –length is specified, then only the first length characters are used in the comparison. If –length is negative, it is ignored. If –nocase is specified, then the strings are compared in a case–insensitive manner.

string equal ?nocase? ?length int? string1 string2
Perform a character–by–character comparison of strings string1 and string2. Returns 1 if string1 and string2 are identical, or 0 when not. If –length is specified, then only the first length characters are used in the comparison. If –length is negative, it is ignored. If –nocase is specified, then the strings are compared in a case–insensitive manner.
**string first string1 string2 ?startIndex?**

Search `string2` for a sequence of characters that exactly match the characters in `string1`. If found, return the index of the first character in the first such match within `string2`. If not found, return −1. If `startIndex` is specified (in any of the forms accepted by the `index` method), then the search is constrained to start with the character in `string2` specified by the index. For example,

```
string first a 0a23456789abcdef 5
```

will return 10, but

```
string first a 0123456789abcdef 11
```

will return −1.

**string index string charIndex**

Returns the `charIndex`'th character of the `string` argument. A `charIndex` of 0 corresponds to the first character of the string. `charIndex` may be specified as follows:

**integer**

The char specified at this integral index.

**end**

The last char of the string.

**end−integer**

The last char of the string minus the specified integer offset (e.g. `end−1` would refer to the "c" in "abcd").

If `charIndex` is less than 0 or greater than or equal to the length of the string then an empty string is returned.

**string is class ?−strict? ?−failindex varname? string**

Returns 1 if `string` is a valid member of the specified character class, otherwise returns 0. If `−strict` is specified, then an empty string returns 0, otherwise and empty string will return 1 on any class. If `−failindex` is specified, then if the function returns 0, the index in the string where the class was no longer valid will be stored in the variable named `varname`. The `varname` will not be set if the function returns 1. The following character classes are recognized (the class name can be abbreviated):

**alnum**

Any Unicode alphabet or digit character.

**alpha**

Any Unicode alphabet character.

**ascii**

Any character with a value less than \u0080 (those that are in the 7–bit ascii range).

**boolean**

Any of the forms allowed to `Tcl_GetBoolean`.

**control**

Any Unicode control character.

**digit**

Any Unicode digit character. Note that this includes characters outside of the [0–9] range.

**double**
Any of the valid forms for a double in Tcl, with optional surrounding whitespace. In case of under/overflow in the value, 0 is returned and the varname will contain −1.

false
Any of the forms allowed to Tcl_GetBoolean where the value is false.

graph
Any Unicode printing character, except space.

integer
Any of the valid forms for an ordinary integer in Tcl, with optional surrounding whitespace. In case of under/overflow in the value, 0 is returned and the varname will contain −1.

lower
Any Unicode lower case alphabet character.

print
Any Unicode printing character, including space.

punct
Any Unicode punctuation character.

space
Any Unicode space character.

true
Any of the forms allowed to Tcl_GetBoolean where the value is true.

upper
Any upper case alphabet character in the Unicode character set.

wordchar
Any Unicode word character. That is any alphanumeric character, and any Unicode connector punctuation characters (e.g. underscore).

xdigit
Any hexadecimal digit character ([0−9A−Fa−f]).

In the case of boolean, true and false, if the function will return 0, then the varname will always be set to 0, due to the varied nature of a valid boolean value.

string last string1 string2 ?lastIndex?
Search string2 for a sequence of characters that exactly match the characters in string1. If found, return the index of the first character in the last such match within string2. If there is no match, then return −1. If lastIndex is specified (in any of the forms accepted by the index method), then only the characters in string2 at or before the specified lastIndex will be considered by the search. For example,

```
string last a 0a23456789abcdef 15
```
will return 10, but

```
string last a 0a23456789abcdef 9
```
will return 1.

string length string
Returns a decimal string giving the number of characters in string. Note that this is not necessarily the same as the number of bytes used to store the string. If the object is a ByteArray object (such as those returned from reading a binary encoded channel), then this will return the actual byte length of the
object.

string map ?–nocase? mapping string
Replaces substrings in string based on the key–value pairs in mapping. mapping is a list of key value key value ... as in the form returned by array get. Each instance of a key in the string will be replaced with its corresponding value. If –nocase is specified, then matching is done without regard to case differences. Both key and value may be multiple characters. Replacement is done in an ordered manner, so the key appearing first in the list will be checked first, and so on. string is only iterated over once, so earlier key replacements will have no affect for later key matches. For example,

```
string map {abc 1 ab 2 a 3 1 0} labcaababcabababc
```

will return the string 01321221.

Note that if an earlier key is a prefix of a later one, it will completely mask the later one. So if the previous example is reordered like this,

```
string map {1 0 ab 2 a 3 abc 1} labcaababcabababc
```

it will return the string 02c322c222c.

string match ?–nocase? pattern string
See if pattern matches string; return 1 if it does, 0 if it doesn't. If –nocase is specified, then the pattern attempts to match against the string in a case insensitive manner. For the two strings to match, their contents must be identical except that the following special sequences may appear in pattern:

* Matches any sequence of characters in string, including a null string.

? Matches any single character in string.

[chars] Matches any character in the set given by chars. If a sequence of the form x–y appears in chars, then any character between x and y, inclusive, will match. When used with –nocase, the end points of the range are converted to lower case first. Whereas {{A–z}} matches '_.' when matching case–sensitively ('.' falls between the 'Z' and 'a'), with –nocase this is considered like {{A–Za–z}} (and probably what was meant in the first place).

\x Matches the single character x. This provides a way of avoiding the special interpretation of the characters *?[] in pattern.

string range string first last
Returns a range of consecutive characters from string, starting with the character whose index is first and ending with the character whose index is last. An index of 0 refers to the first character of the string. first and last may be specified as for the index method. If first is less than zero then it is treated as if it were zero, and if last is greater than or equal to the length of the string then it is treated as if it were end. If first is greater than last then an empty string is returned.

string repeat string count
Returns string repeated count number of times.

string replace string first last ?newstring?

Removes a range of consecutive characters from string, starting with the character whose index is first and ending with the character whose index is last. An index of 0 refers to the first character of the string. First and last may be specified as for the index method. If newstring is specified, then it is placed in the removed character range. If first is less than zero then it is treated as if it were zero, and if last is greater than or equal to the length of the string then it is treated as if it were end. If first is greater than last or the length of the initial string, or last is less than 0, then the initial string is returned untouched.

string tolower string ?first? ?last?
Returns a value equal to string except that all upper (or title) case letters have been converted to lower case. If first is specified, it refers to the first char index in the string to start modifying. If last is specified, it refers to the char index in the string to stop at (inclusive). first and last may be specified as for the index method.

string totitle string ?first? ?last?
Returns a value equal to string except that the first character in string is converted to its Unicode title case variant (or upper case if there is no title case variant) and the rest of the string is converted to lower case. If first is specified, it refers to the first char index in the string to start modifying. If last is specified, it refers to the char index in the string to stop at (inclusive). first and last may be specified as for the index method.

string toupper string ?first? ?last?
Returns a value equal to string except that all lower (or title) case letters have been converted to upper case. If first is specified, it refers to the first char index in the string to start modifying. If last is specified, it refers to the char index in the string to stop at (inclusive). first and last may be specified as for the index method.

string trim string ?chars?
Returns a value equal to string except that any leading or trailing characters from the set given by chars are removed. If chars is not specified then white space is removed (spaces, tabs, newlines, and carriage returns).

string trimleft string ?chars?
Returns a value equal to string except that any leading characters from the set given by chars are removed. If chars is not specified then white space is removed (spaces, tabs, newlines, and carriage returns).

string trimright string ?chars?
Returns a value equal to string except that any trailing characters from the set given by chars are removed. If chars is not specified then white space is removed (spaces, tabs, newlines, and carriage returns).

string wordend string charIndex
Returns the index of the character just after the last one in the word containing character charIndex of string. charIndex may be specified as for the index method. A word is considered to be any contiguous range of alphanumeric (Unicode letters or decimal digits) or underscore (Unicode connector punctuation) characters, or any single character other than these.

string wordstart string charIndex
Returns the index of the first character in the word containing character charIndex of string. charIndex may be specified as for the index method. A word is considered to be any contiguous range of alphanumeric (Unicode letters or decimal digits) or underscore (Unicode connector punctuation) characters, or any single character other than these.
EXAMPLE

Test if the string in the variable *string* is a proper non-empty prefix of the string *foobar*.

```tcl
set length [string length $string]
if {$length == 0} {
    set isPrefix 0
} else {
    set isPrefix [string equal -length $string $string "foobar"]
}
```

SEE ALSO

`expr`, `list`

KEYWORDS

`case conversion`, `compare`, `index`, `match`, `pattern`, `string`, `word`, `equal`, `ctype`

Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
Copyright © 1995-1997 Roger E. Critchlow Jr.
NAME

subst – Perform backslash, command, and variable substitutions

SYNOPSIS


DESCRIPTION

This command performs variable substitutions, command substitutions, and backslash substitutions on its string argument and returns the fully–substituted result. The substitutions are performed in exactly the same way as for Tcl commands. As a result, the string argument is actually substituted twice, once by the Tcl parser in the usual fashion for Tcl commands, and again by the subst command.

If any of the −nobackslashes, −nocommands, or −novariables are specified, then the corresponding substitutions are not performed. For example, if −nocommands is specified, command substitution is not performed: open and close brackets are treated as ordinary characters with no special interpretation.

Note that the substitution of one kind can include substitution of other kinds. For example, even when the −novariables option is specified, command substitution is performed without restriction. This means that any variable substitution necessary to complete the command substitution will still take place. Likewise, any command substitution necessary to complete a variable substitution will take place, even when −nocommands is specified. See the EXAMPLES below.

If an error occurs during substitution, then subst will return that error. If a break exception occurs during command or variable substitution, the result of the whole substitution will be the string (as substituted) up to the start of the substitution that raised the exception. If a continue exception occurs during the evaluation of a command or variable substitution, an empty string will be substituted for that entire command or variable substitution (as long as it is well–formed Tcl.) If a return exception occurs, or any other return code is returned during command or variable substitution, then the returned value is substituted for that substitution. See the EXAMPLES below. In this way, all exceptional return codes are ``caught" by subst. The subst command itself will either return an error, or will complete successfully.

EXAMPLES

When it performs its substitutions, subst does not give any special treatment to double quotes or curly braces (except within command substitutions) so the script

set a 44
subst {xyz {$a}}
returns \texttt{``xyz \{44\}''}, not \texttt{``xyz \{$a\}''} and the script

\begin{verbatim}
set a "p\} q \{r"
subst \{xyz \{$a\}\}
\end{verbatim}

return \texttt{``xyz \{p} q \{r\}''}, not \texttt{``xyz \{p\} q \{r\}''}. When command substitution is performed, it includes any variable substitution necessary to evaluate the script.

\begin{verbatim}
set a 44
subst \-novariables \{$a \[format \$a\]\}
\end{verbatim}

returns \texttt{``$a 44''}, not \texttt{``$a $a''}. Similarly, when variable substitution is performed, it includes any command substitution necessary to retrieve the value of the variable.

\begin{verbatim}
proc b {} {return c}
array set a \{c c \[b\] tricky\}
subst \-nocommands \{\[b\] \$a(\[b\])\}
\end{verbatim}

returns \texttt{``\[b\] c''}, not \texttt{``\[b\] tricky''}. The continue and break exceptions allow command substitutions to prevent substitution of the rest of the command substitution and the rest of \texttt{string} respectively, giving script authors more options when processing text using \texttt{subst}. For example, the script

\begin{verbatim}
subst \{abc,\[break\],def\}
\end{verbatim}

returns \texttt{``abc''}, not \texttt{``abc,def''} and the script

\begin{verbatim}
subst \{abc,\[continue;expr 1+2\],def\}
\end{verbatim}

returns \texttt{``abc,def''}, not \texttt{``abc,3,def''}. Other exceptional return codes substitute the returned value

\begin{verbatim}
subst \{abc,\[return foo;expr 1+2\],def\}
\end{verbatim}

returns \texttt{``abc,foo,def''}, not \texttt{``abc,3,def''} and

\begin{verbatim}
subst \{abc,\[return \-code 10 foo;expr 1+2\],def\}
\end{verbatim}

also returns \texttt{``abc,foo,def''}, not \texttt{``abc,3,def''}.
SEE ALSO

Tcl, eval, break, continue

KEYWORDS

backslash substitution, command substitution, variable substitution

Copyright ' 1994 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 2001 Donal K. Fellows
Copyright ' 1995-1997 Roger E. Critchlow Jr.
switch

NAME

switch – Evaluate one of several scripts, depending on a given value

SYNOPSIS

switch ?options? string pattern body ?pattern body ...?
switch ?options? string {pattern body ?pattern body ...?}

DESCRIPTION

- exact
- glob
- regexp

==

EXAMPLES

SEE ALSO

KEYWORDS

NAME

switch – Evaluate one of several scripts, depending on a given value

SYNOPSIS

switch ?options? string pattern body ?pattern body ...?
switch ?options? string {pattern body ?pattern body ...?}

DESCRIPTION

The switch command matches its string argument against each of the pattern arguments in order. As soon as it finds a pattern that matches string it evaluates the following body argument by passing it recursively to the Tcl interpreter and returns the result of that evaluation. If the last pattern argument is default then it matches anything. If no pattern argument matches string and no default is given, then the switch command returns an empty string.

If the initial arguments to switch start with – then they are treated as options. The following options are currently supported:

-exact
    Use exact matching when comparing string to a pattern. This is the default.
-glob
    When matching string to the patterns, use glob-style matching (i.e. the same as implemented by the string match command).
-regexp

switch
When matching string to the patterns, use regular expression matching (as described in the `re_syntax` reference page).

Marks the end of options. The argument following this one will be treated as string even if it starts with a −.

Two syntaxes are provided for the pattern and body arguments. The first uses a separate argument for each of the patterns and commands; this form is convenient if substitutions are desired on some of the patterns or commands. The second form places all of the patterns and commands together into a single argument; the argument must have proper list structure, with the elements of the list being the patterns and commands. The second form makes it easy to construct multi–line switch commands, since the braces around the whole list make it unnecessary to include a backslash at the end of each line. Since the pattern arguments are in braces in the second form, no command or variable substitutions are performed on them; this makes the behavior of the second form different than the first form in some cases.

If a body is specified as ```−''` it means that the body for the next pattern should also be used as the body for this pattern (if the next pattern also has a body of ```−''` then the body after that is used, and so on). This feature makes it possible to share a single body among several patterns.

Beware of how you place comments in switch commands. Comments should only be placed inside the execution body of one of the patterns, and not intermingled with the patterns.

EXAMPLES

The switch command can match against variables and not just literals, as shown here (the result is 2):

```
set foo "abc"
switch abc a − b {expr 1} $foo {expr 2} default {expr 3}
```

Using glob matching and the fall−through body is an alternative to writing regular expressions with alternations, as can be seen here (this returns 1):

```
switch −glob aaab {
    a*b     −
    b       {expr 1}
    a*      {expr 2}
    default {expr 3}
}
```

Whenever nothing matches, the default clause (which must be last) is taken. This example has a result of 3:

```
switch xyz {
    a −
    b {
        # Correct Comment Placement
        expr 1
    }
    c {
```

```
expr 2
}
default {
    expr 3
}
}

SEE ALSO

for, if, regexp

KEYWORDS

switch, match, regular expression

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1997 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

tcltest – Test harness support code and utilities

SYNOPSIS

package require tcltest ?2.2.5?
tcltest::test name description ?option value ...?
tcltest::test name description ?constraints? body result
tcltest::loadTestedCommands
tcltest::makeDirectory name ?directory?
tcltest::removeDirectory name ?directory?
tcltest::makeFile contents name ?directory?
tcltest::removeFile name ?directory?
tcltest::viewFile name ?directory?
tcltest::cleanupTests ?runningMultipleTests?
tcltest::runAllTests
tcltest::configure
tcltest::configure option
tcltest::configure option value ?option value ...?
tcltest::customMatch mode command
tcltest::testConstraint constraint ?value?
tcltest::outputChannel ?channelID?
tcltest::errorChannel ?channelID?
tcltest::interpreter ?interp?
tcltest::debug ?level?
tcltest::errorFile ?filename?
tcltest::limitConstraints ?boolean?
tcltest::loadFile ?filename?
tcltest::loadScript ?script?
tcltest::match ?patternList?
tcltest::matchDirectories ?patternList?
tcltest::matchFiles ?patternList?
tcltest::outputFile ?filename?
tcltest::preserveCore ?level?
tcltest::singleProcess ?boolean?
tcltest::skip ?patternList?
tcltest::skipDirectories ?patternList?
tcltest::skipFiles ?patternList?
tcltest::temporaryDirectory ?directory?
tcltest::testsDirectory ?directory?
tcltest::verbose ?level?
tcltest::test name description optionList
tcltest::bytestring string
DESCRIPTION

COMMANDS

test name description ?option value ...?
test name description ?constraints? body result
loadTestedCommands
makeFile contents name ?directory?
removeFile name ?directory?
makeDirectory name ?directory?
removeDirectory name ?directory?
viewFile file ?directory?
cleanupTests
runAllTests

CONFIGURATION COMMANDS

configure
configure option
configure option value ?option value ...?
customMatch mode script
testConstraint constraint ?boolean?
interpreter ?executableName?
outputChannel ?channelID?
errorChannel ?channelID?

SHORTCUT COMMANDS

debug ?level?
errorFile ?filename?
limitConstraints ?boolean?
loadFile ?filename?
loadScript ?script?
match ?patternList?
matchDirectories ?patternList?
matchFiles ?patternList?
outputFile ?filename?
preserveCore ?level?
singleProcess ?boolean?
skip ?patternList?
skipDirectories ?patternList?
skipFiles ?patternList?
temporaryDirectory ?directory?
testsDirectory ?directory?
verbose ?level?

OTHER COMMANDS

test name description optionList
workingDirectory ?directoryName?
normalizeMsg msg
normalizePath pathVar
bytestring string

TESTS
  -constraints keywordList|expression
  -setup script
  -body script
  -cleanup script
  -match mode
  -result expectedValue
  -output expectedValue
  -errorOutput expectedValue
  -returnCodes expectedCodeList

TEST CONSTRAINTS
  singleTestInterp
  unix
  win
  nt
  95
  98
  mac
  unixOrWin
  macOrWin
  macOrUnix
  tempNotWin
  tempNotMac
  unixCrash
  winCrash
  macCrash
  emptyTest
  knownBug
  nonPortable
  userInteraction
  interactive
  nonBlockFiles
  asyncPipeClose
  unixExecs
  hasIsoLocale
  root
  notRoot
  eformat
 stdio

RUNNING ALL TESTS
CONFIGURABLE OPTIONS
  --singleproc boolean
-debug level
  0
  1
  2
  3
-verbose level
  body (b)
  pass (p)
  skip (s)
  start (t)
  error (e)
-preservecore level
  0
  1
  2
-limitconstraints boolean
-constraints list
-tempdir directory
-testdir directory
-file patternList
-notfile patternList
-relateddir patternList
-asidefromdir patternList
-match patternList
-skip patternList
-load script
-loadfile filename
-outfile filename
-errfile filename

CREATING TEST SUITES WITH TCLTEST
COMPATIBILITY
KNOWN ISSUES
KEYWORDS

NAME

tcltest – Test harness support code and utilities

SYNOPSIS

package require tcltest ?2.2.5?
tcltest::test name description ?option value ...?
tcltest::test name description ?constraints? body result
tcltest::loadTestedCommands

tcltest::makeDirectory name ?directory?
DESCRIPTION

The tcltest package provides several utility commands useful in the construction of test suites for code instrumented to be run by evaluation of Tcl commands. Notably the built-in commands of the Tcl library itself are tested by a test suite using the tcltest package.

All the commands provided by the tcltest package are defined in and exported from the ::tcltest namespace, as indicated in the SYNOPSIS above. In the following sections, all commands will be described by their
simple names, in the interest of brevity.

The central command of `tcltest` is `[test]` that defines and runs a test. Testing with `[test]` involves evaluation of a Tcl script and comparing the result to an expected result, as configured and controlled by a number of options. Several other commands provided by `tcltest` govern the configuration of `[test]` and the collection of many `[test]` commands into test suites.

See CREATING TEST SUITES WITH TCLTEST below for an extended example of how to use the commands of `tcltest` to produce test suites for your Tcl–enabled code.

**COMMANDS**

`test name description ?option value ...?`

Defines and possibly runs a test with the name `name` and description `description`. The name and description of a test are used in messages reported by `[test]` during the test, as configured by the options of `tcltest`. The remaining `option value` arguments to `[test]` define the test, including the scripts to run, the conditions under which to run them, the expected result, and the means by which the expected and actual results should be compared. See TESTS below for a complete description of the valid options and how they define a test. The `[test]` command returns an empty string.

`test name description ?constraints? body result`

This form of `[test]` is provided to support test suites written for version 1 of the `tcltest` package, and also a simpler interface for a common usage. It is the same as `[test name description −constraints constraints −body body −result result]`. All other options to `[test]` take their default values. When `constraints` is omitted, this form of `[test]` can be distinguished from the first because all options begin with `−−`.

`loadTestedCommands`

Evaluates in the caller’s context the script specified by `[configure −load]` or `[configure −loadfile]`. Returns the result of that script evaluation, including any error raised by the script. Use this command and the related configuration options to provide the commands to be tested to the interpreter running the test suite.

`makeFile contents name ?directory?`

Creates a file named `name` relative to directory `directory` and write `contents` to that file using the encoding `[encoding system]`. If `contents` does not end with a newline, a newline will be appended so that the file named `name` does end with a newline. Because the system encoding is used, this command is only suitable for making text files. The file will be removed by the next evaluation of `[cleanupTests]`, unless it is removed by `[removeFile]` first. The default value of `directory` is the directory `[configure −tmpdir]`. Returns the full path of the file created. Use this command to create any text file required by a test with contents as needed.

`removeFile name ?directory?`

Forces the file referenced by `name` to be removed. This file name should be relative to `directory`. The default value of `directory` is the directory `[configure −tmpdir]`. Returns an empty string. Use this command to delete files created by `[makeFile]`.

`makeDirectory name ?directory?`

Creates a directory named `name` relative to directory `directory`. The directory will be removed by the next evaluation of `[cleanupTests]`, unless it is removed by `[removeDirectory]` first. The default value
of directory is the directory [configure −tmpdir]. Returns the full path of the directory created. Use this command to create any directories that are required to exist by a test.

**removeDirectory**

name ?directory?

Forces the directory referenced by name to be removed. This directory should be relative to directory. The default value of directory is the directory [configure −tmpdir]. Returns an empty string. Use this command to delete any directories created by [makeDirectory].

**viewFile**

file ?directory?

Returns the contents of file, except for any final newline, just as [read −nonewline] would return. This file name should be relative to directory. The default value of directory is the directory [configure −tmpdir]. Use this command as a convenient way to turn the contents of a file generated by a test into the result of that test for matching against an expected result. The contents of the file are read using the system encoding, so its usefulness is limited to text files.

**cleanupTests**

Intended to clean up and summarize after several tests have been run. Typically called once per test file, at the end of the file after all tests have been completed. For best effectiveness, be sure that the [cleanupTests] is evaluated even if an error occurs earlier in the test file evaluation.

Prints statistics about the tests run and removes files that were created by [makeDirectory] and [makeFile] since the last [cleanupTests]. Names of files and directories in the directory [configure −tmpdir] created since the last [cleanupTests], but not created by [makeFile] or [makeDirectory] are printed to [outputChannel]. This command also restores the original shell environment, as described by the ::env array. Returns an empty string.

**runAllTests**

This is a master command meant to run an entire suite of tests, spanning multiple files and/or directories, as governed by the configurable options of tcltest. See RUNNING ALL TESTS below for a complete description of the many variations possible with [runAllTests].

### CONFIGURATION COMMANDS

**configure**

Returns the list of configurable options supported by tcltest. See CONFIGURABLE OPTIONS below for the full list of options, their valid values, and their effect on tcltest operations.

**configure option**

Returns the current value of the supported configurable option option. Raises an error if option is not a supported configurable option.

**configure option value ?option value ...?**

Sets the value of each configurable option option to the corresponding value value, in order. Raises an error if an option is not a supported configurable option, or if value is not a valid value for the corresponding option, or if a value is not provided. When an error is raised, the operation of [configure] is halted, and subsequent option value arguments are not processed.

If the environment variable ::env(TCLTEST_OPTIONS) exists when the tcltest package is loaded (by [package require tcltest]) then its value is taken as a list of arguments to pass to [configure]. This allows the default values of the configuration options to be set by the environment.

**customMatch**

mode script
Registers *mode* as a new legal value of the *−match* option to [*test*]. When the *−match mode* option is passed to [*test*], the script *script* will be evaluated to compare the actual result of evaluating the body of the test to the expected result. To perform the match, the *script* is completed with two additional words, the expected result, and the actual result, and the completed script is evaluated in the global namespace. The completed script is expected to return a boolean value indicating whether or not the results match. The built−in matching modes of [*test*] are *exact*, *glob*, and *regexp*.

**testConstraint** *constraint ?boolean?*

Sets or returns the boolean value associated with the named *constraint*. See TEST CONSTRAINTS below for more information.

**interpreter ?executableName?**

Sets or returns the name of the executable to be [exec]ed by [runAllTests] to run each test file when [*configure −singleproc*] is false. The default value for [interpreter] is the name of the currently running program as returned by [info nameofexecutable].

**outputChannel ?channelID?**

Sets or returns the output channel ID. This defaults to stdout. Any test that prints test related output should send that output to [outputChannel] rather than letting that output default to stdout.

**errorChannel ?channelID?**

Sets or returns the error channel ID. This defaults to stderr. Any test that prints error messages should send that output to [errorChannel] rather than printing directly to stderr.

## SHORTCUT COMMANDS

**debug ?level?**

Same as [*configure −debug ?level?*].

**errorMessage ?filename?**

Same as [*configure −errfile ?filename?*].

**limitConstraints ?boolean?**

Same as [*configure −limitconstraints ?boolean?*].

**loadFile ?filename?**

Same as [*configure −loadfile ?filename?*].

**loadScript ?script?**

Same as [*configure −load ?script?*].

**match ?patternList?**

Same as [*configure −match ?patternList?*].

**matchDirectories ?patternList?**

Same as [*configure −relateddir ?patternList?*].

**matchFiles ?patternList?**

Same as [*configure −file ?patternList?*].

**outputFile ?filename?**

Same as [*configure −outfile ?filename?*].

**preserveCore ?level?**

Same as [*configure −preservecore ?level?*].

**singleProcess ?boolean?**

Same as [*configure −singleproc ?boolean?*].

**skip ?patternList?**
same as \[configure \--skip ?patternList?\].

**skipDirectories** ?patternList?

same as \[configure \--asidewithdir ?patternList?\].

**skipFiles** ?patternList?

same as \[configure \--notfromdir ?patternList?\].

**temporaryDirectory** ?directory?

same as \[configure \--tmpdir ?directory?\].

**testsDirectory** ?directory?

same as \[configure \--testdir ?directory?\].

**verbose** ?level?

same as \[configure \--verbose ?level?\].

**OTHER COMMANDS**

The remaining commands provided by **tcltest** have better alternatives provided by **tcltest** or **Tcl** itself. They are retained to support existing test suites, but should be avoided in new code.

**test name description optionList**

This form of \[test\] was provided to enable passing many options spanning several lines to \[test\] as a single argument quoted by braces, rather than needing to backslash quote the newlines between arguments to \[test\]. The **optionList** argument is expected to be a list with an even number of elements representing **option** and **value** arguments to pass to \[test\]. However, these values are not passed directly, as in the alternate forms of **switch**. Instead, this form makes an unfortunate attempt to overthrow Tcl's substitution rules by performing substitutions on some of the list elements as an attempt to implement a "do what I mean" interpretation of a brace-enclosed "block". The result is nearly impossible to document clearly, and for that reason this form is not recommended. See the examples in **CREATING TEST SUITES WITH TCLTEST** below to see that this form is really not necessary to avoid backslash-quoted newlines. If you insist on using this form, examine the source code of **tcltest** if you want to know the substitution details, or just enclose the third through last argument to \[test\] in braces and hope for the best.

**workingDirectory** ?directoryName?

Sets or returns the current working directory when the test suite is running. The default value for workingDirectory is the directory in which the test suite was launched. The Tcl commands \[cd\] and \[pwd\] are sufficient replacements.

**normalizeMsg** msg

Returns the result of removing the "extra" newlines from msg, where "extra" is rather imprecise. Tcl offers plenty of string processing commands to modify strings as you wish, and \[customMatch\] allows flexible matching of actual and expected results.

**normalizePath** pathVar

Resolves symlinks in a path, thus creating a path without internal redirection. It is assumed that pathVar is absolute. pathVar is modified in place. The Tcl command \[file normalize\] is a sufficient replacement.

**bytestring** string

Construct a string that consists of the requested sequence of bytes, as opposed to a string of properly formed UTF-8 characters using the value supplied in string. This allows the tester to create
denormalized or improperly formed strings to pass to C procedures that are supposed to accept strings with embedded NULL types and confirm that a string result has a certain pattern of bytes. This is exactly equivalent to the Tcl command [encoding convertfrom identity].

TESTS

The [test] command is the heart of the tcltest package. Its essential function is to evaluate a Tcl script and compare the result with an expected result. The options of [test] define the test script, the environment in which to evaluate it, the expected result, and how to compare the actual result to the expected result. Some configuration options of tcltest also influence how [test] operates.

The valid options for [test] are summarized:

```
test name description
    ?−constraints keywordList|expression?
    ?−setup script?
    ?−body script?
    ?−cleanup script?
    ?−result expectedAnswer?
    ?−output expectedOutput?
    ?−errorOutput expectedError?
    ?−returnCodes codeList?
    ?−match mode?
```

The name may be any string. It is conventional to choose a name according to the pattern:

```
target−majorNum.minorNum
```

For white-box (regression) tests, the target should be the name of the C function or Tcl procedure being tested. For black-box tests, the target should be the name of the feature being tested. Some conventions call for the names of black-box tests to have the suffix _bb. Related tests should share a major number. As a test suite evolves, it is best to have the same test name continue to correspond to the same test, so that it remains meaningful to say things like ``Test foo−1.3 passed in all releases up to 3.4, but began failing in release 3.5.''

During evaluation of [test], the name will be compared to the lists of string matching patterns returned by [configure −match], and [configure −skip]. The test will be run only if name matches any of the patterns from [configure −match] and matches none of the patterns from [configure −skip].

The description should be a short textual description of the test. The description is included in output produced by the test, typically test failure messages. Good description values should briefly explain the purpose of the test to users of a test suite. The name of a Tcl or C function being tested should be included in the description for regression tests. If the test case exists to reproduce a bug, include the bug ID in the description.

Valid attributes and associated values are:

```
−constraints keywordList|expression
```
The optional --constraints attribute can be list of one or more keywords or an expression. If the
--constraints value is a list of keywords, each of these keywords should be the name of a constraint
defined by a call to [testConstraint]. If any of the listed constraints is false or does not exist, the test
is skipped. If the --constraints value is an expression, that expression is evaluated. If the expression
evaluates to true, then the test is run. Note that the expression form of --constraints may interfere
with the operation of [configure --constraints] and [configure --limitconstraints], and is not
recommended. Appropriate constraints should be added to any tests that should not always be run.
That is, conditional evaluation of a test should be accomplished by the --constraints option, not by
conditional evaluation of [test]. In that way, the same number of tests are always reported by the test
suite, though the number skipped may change based on the testing environment. The default value is
an empty list. See TEST CONSTRAINTS below for a list of built-in constraints and information on
how to add your own constraints.

--setup script
The optional --setup attribute indicates a script that will be run before the script indicated by the
--body attribute. If evaluation of script raises an error, the test will fail. The default value is an empty
script.

--body script
The --body attribute indicates the script to run to carry out the test. It must return a result that can be
checked for correctness. If evaluation of script raises an error, the test will fail. The default value is an empty script.

--cleanup script
The optional --cleanup attribute indicates a script that will be run after the script indicated by the
--body attribute. If evaluation of script raises an error, the test will fail. The default value is an empty script.

--match mode
The --match attribute determines how expected answers supplied by --result, --output, and
--errorOutput are compared. Valid values for mode are regexp, glob, exact, and any value registered
by a prior call to [customMatch]. The default value is exact.

--result expectedValue
The --result attribute supplies the expectedValue against which the return value from script will be
compared. The default value is an empty string.

--output expectedValue
The --output attribute supplies the expectedValue against which any output sent to stdout or
[outputChannel] during evaluation of the script(s) will be compared. Note that only output printed
using [:puts] is used for comparison. If --output is not specified, output sent to stdout and
[outputChannel] is not processed for comparison.

--errorOutput expectedValue
The --errorOutput attribute supplies the expectedValue against which any output sent to stderr or
[errorChannel] during evaluation of the script(s) will be compared. Note that only output printed
using [:puts] is used for comparison. If --errorOutput is not specified, output sent to stderr and
[errorChannel] is not processed for comparison.

--returnCodes expectedCodeList
The optional --returnCodes attribute supplies expectedCodeList, a list of return codes that may be
accepted from evaluation of the --body script. If evaluation of the --body script returns a code not in
the expectedCodeList, the test fails. All return codes known to [return], in both numeric and symbolic
form, including extended return codes, are acceptable elements in the expectedCodeList. Default value is \{ok return\}.

To pass, a test must successfully evaluate its \−setup, \−body, and \−cleanup\ scripts. The return code of the \−body\ script and its result must match expected values, and if specified, output and error data from the test must match expected \−output\ and \−errorOutput\ values. If any of these conditions are not met, then the test fails. Note that all scripts are evaluated in the context of the caller of \[test\].

As long as \[test\] is called with valid syntax and legal values for all attributes, it will not raise an error. Test failures are instead reported as output written to \[outputChannel\]. In default operation, a successful test produces no output. The output messages produced by \[test\] are controlled by the \[configure \−verbose\] option as described in CONFIGURABLE OPTIONS below. Any output produced by the test scripts themselves should be produced using \[::puts\] to \[outputChannel\] or \[errorChannel\], so that users of the test suite may easily capture output with the \[configure \−outfile\] and \[configure \−errfile\] options, and so that the \−output\ and \−errorOutput\ attributes work properly.

**TEST CONSTRAINTS**

Constraints are used to determine whether or not a test should be skipped. Each constraint has a name, which may be any string, and a boolean value. Each \[test\] has a \−constraints\ value which is a list of constraint names. There are two modes of constraint control. Most frequently, the default mode is used, indicated by a setting of \[configure \−limitconstraints\] to false. The test will run only if all constraints in the list are true−valued. Thus, the \−constraints\ option of \[test\] is a convenient, symbolic way to define any conditions required for the test to be possible or meaningful. For example, a \[test\] with \−constraints unix\ will only be run if the constraint unix is true, which indicates the test suite is being run on a Unix platform.

Each \[test\] should include whatever \−constraints\ are required to constrain it to run only where appropriate. Several constraints are pre−defined in the tcltest package, listed below. The registration of user−defined constraints is performed by the \[testConstraint\] command. User−defined constraints may appear within a test file, or within the script specified by the \[configure \−load\] or \[configure \−loadfile\] options.

The following is a list of constraints pre−defined by the tcltest package itself:

- \*singleTestInterp*: test can only be run if all test files are sourced into a single interpreter
- \*unix*: test can only be run on any Unix platform
- \*win*: test can only be run on any Windows platform
- \*nt*: test can only be run on any Windows NT platform
- \*95*: test can only be run on any Windows 95 platform
- \*98*: test can only be run on any Windows 98 platform
mac  
  test can only be run on any Mac platform

unixOrWin  
  test can only be run on a Unix or Windows platform

macOrWin  
  test can only be run on a Mac or Windows platform

macOrUnix  
  test can only be run on a Mac or Unix platform

tempNotWin  
  test can not be run on Windows. This flag is used to temporarily disable a test.

tempNotMac  
  test can not be run on a Mac. This flag is used to temporarily disable a test.

unixCrash  
  test crashes if it's run on Unix. This flag is used to temporarily disable a test.

winCrash  
  test crashes if it's run on Windows. This flag is used to temporarily disable a test.

macCrash  
  test crashes if it's run on a Mac. This flag is used to temporarily disable a test.

emptyTest  
  test is empty, and so not worth running, but it remains as a place−holder for a test to be written in the future. This constraint has value false to cause tests to be skipped unless the user specifies otherwise.

knownBug  
  test is known to fail and the bug is not yet fixed. This constraint has value false to cause tests to be skipped unless the user specifies otherwise.

nonPortable  
  test can only be run in some known development environment. Some tests are inherently non−portable because they depend on things like word length, file system configuration, window manager, etc. This constraint has value false to cause tests to be skipped unless the user specifies otherwise.

userInteraction  
  test requires interaction from the user. This constraint has value false to causes tests to be skipped unless the user specifies otherwise.

interactive  
  test can only be run if the interpreter is in interactive mode (when the global tcl_interactive variable is set to 1).

nonBlockFiles  
  test can only be run if platform supports setting files into nonblocking mode

asyncPipeClose  
  test can only be run if platform supports async flush and async close on a pipe

unixExecs  
  test can only be run if this machine has Unix−style commands cat, echo, sh, wc, rm, sleep, fgrep, ps, chmod, and mkdir available

hasIsoLocale  
  test can only be run if can switch to an ISO locale

root
test can only run if Unix user is root

notRoot
test can only run if Unix user is not root
eformat
test can only run if app has a working version of sprintf with respect to the "e" format of floating-point numbers.
stdio
test can only be run if [interpreter] can be [open]ed as a pipe.

The alternative mode of constraint control is enabled by setting [configure −limitconstraints] to true. With that configuration setting, all existing constraints other than those in the constraint list returned by [configure −constraints] are set to false. When the value of [configure −constraints] is set, all those constraints are set to true. The effect is that when both options [configure −constraints] and [configure −limitconstraints] are in use, only those tests including only constraints from the [configure −constraints] list are run; all others are skipped. For example, one might set up a configuration with

```
configure −constraints knownBug \
 −limitconstraints true \
 −verbose pass
```

to run exactly those tests that exercise known bugs, and discover whether any of them pass, indicating the bug had been fixed.

RUNNING ALL TESTS

The single command [runAllTests] is evaluated to run an entire test suite, spanning many files and directories. The configuration options of tcltest control the precise operations. The [runAllTests] command begins by printing a summary of its configuration to [outputChannel].

Test files to be evaluated are sought in the directory [configure −testdir]. The list of files in that directory that match any of the patterns in [configure −file] and match none of the patterns in [configure −notfile] is generated and sorted. Then each file will be evaluated in turn. If [configure −singleproc] is true, then each file will be [source]d in the caller's context. If it is false, then a copy of [interpreter] will be [exec]d to evaluate each file. The multi-process operation is useful when testing can cause errors so severe that a process terminates. Although such an error may terminate a child process evaluating one file, the master process can continue with the rest of the test suite. In multi-process operation, the configuration of tcltest in the master process is passed to the child processes as command line arguments, with the exception of [configure −outfile]. The [runAllTests] command in the master process collects all output from the child processes and collates their results into one master report. Any reports of individual test failures, or messages requested by a [configure −verbose] setting are passed directly on to [outputChannel] by the master process.

After evaluating all selected test files, a summary of the results is printed to [outputChannel]. The summary includes the total number of [test]s evaluated, broken down into those skipped, those passed, and those failed. The summary also notes the number of files evaluated, and the names of any files with failing tests or errors. A list of the constraints that caused tests to be skipped, and the number of tests skipped for each is also printed. Also, messages are printed if it appears that evaluation of a test file has caused any temporary files to
be left behind in [configure −tmpdir].

Having completed and summarized all selected test files, [runAllTests] then recursively acts on subdirectories of [configure −testdir]. All subdirectories that match any of the patterns in [configure −relateddir] and do not match any of the patterns in [configure −asidefromdir] are examined. If a file named all.tcl is found in such a directory, it will be [source]d in the caller's context. Whether or not an examined directory contains an all.tcl file, its subdirectories are also scanned against the [configure −relateddir] and [configure −asidefromdir] patterns. In this way, many directories in a directory tree can have all their test files evaluated by a single [runAllTests] command.

**CONFIGURABLE OPTIONS**

The [configure] command is used to set and query the configurable options of tcltest. The valid options are:

--- singleproc boolean

Controls whether or not [runAllTests] spawns a child process for each test file. No spawning when boolean is true. Default value is false.

--- debug level

Sets the debug level to level, an integer value indicating how much debugging information should be printed to stdout. Note that debug messages always go to stdout, independent of the value of [configure −outfile]. Default value is 0. Levels are defined as:

0  Do not display any debug information.
1  Display information regarding whether a test is skipped because it doesn't match any of the tests that were specified using by [configure −match] (userSpecifiedNonMatch) or matches any of the tests specified by [configure −skip] (userSpecifiedSkip). Also print warnings about possible lack of cleanup or balance in test files. Also print warnings about any re-use of test names.
2  Display the flag array parsed by the command line processor, the contents of the ::env array, and all user–defined variables that exist in the current namespace as they are used.
3  Display information regarding what individual procs in the test harness are doing.

--- verbose level

Sets the type of output verbosity desired to level, a list of zero or more of the elements body, pass, skip, start, and error. Default value is {body error}. Levels are defined as:

body (b)

Display the body of failed tests

pass (p)

Print output when a test passes

skip (s)

Print output when a test is skipped
start (t)
Print output whenever a test starts

error (e)
Print errorInfo and errorCode, if they exist, when a test return code does not match its expected return code

The single letter abbreviations noted above are also recognized so that \texttt{configure -verbose pt} is the same as \texttt{configure -verbose \{pass start\}}.

\texttt{--preservecore level}
Sets the core preservation level to \texttt{level}. This level determines how stringent checks for core files are. Default value is 0. Levels are defined as:

0
No checking – do not check for core files at the end of each test command, but do check for them in \texttt{runAllTests} after all test files have been evaluated.

1
Also check for core files at the end of each \texttt{test} command.

2
Check for core files at all times described above, and save a copy of each core file produced in \texttt{configure -tmpdir}.

\texttt{--limitconstraints boolean}
Sets the mode by which \texttt{test} honors constraints as described in TESTS above. Default value is false.

\texttt{--constraints list}
Sets all the constraints in \texttt{list} to true. Also used in combination with \texttt{configure --limitconstraints true} to control an alternative constraint mode as described in TESTS above. Default value is an empty list.

\texttt{--tmpdir directory}
Sets the temporary directory to be used by \texttt{makeFile}, \texttt{makeDirectory}, \texttt{viewFile}, \texttt{removeFile}, and \texttt{removeDirectory} as the default directory where temporary files and directories created by test files should be created. Default value is \texttt{workingDirectory}.

\texttt{--testdir directory}
Sets the directory searched by \texttt{runAllTests} for test files and subdirectories. Default value is \texttt{workingDirectory}.

\texttt{--file patternList}
Sets the list of patterns used by \texttt{runAllTests} to determine what test files to evaluate. Default value is \texttt{*.*.test}.

\texttt{--notfile patternList}
Sets the list of patterns used by \texttt{runAllTests} to determine what test files to skip. Default value is \texttt{l.*.test}, so that any SCCS lock files are skipped.

\texttt{--relateddir patternList}
Sets the list of patterns used by \texttt{runAllTests} to determine what subdirectories to search for an \texttt{all.tcl} file. Default value is \texttt{*}.

\texttt{--asidefromdir patternList}
Sets the list of patterns used by \texttt{runAllTests} to determine what subdirectories to skip when searching for an \texttt{all.tcl} file. Default value is an empty list.

\texttt{--match patternList}
Set the list of patterns used by [test] to determine whether a test should be run. Default value is *.

−skip patternList
Set the list of patterns used by [test] to determine whether a test should be skipped. Default value is an empty list.

−load script
Sets a script to be evaluated by [loadTestedCommands]. Default value is an empty script.

−loadfile filename
Sets the filename from which to read a script to be evaluated by [loadTestedCommands]. This is an alternative to −load. They cannot be used together.

−outfile filename
Sets the file to which all output produced by tcltest should be written. A file named filename will be [open]ed for writing, and the resulting channel will be set as the value of [outputChannel].

−errfile filename
Sets the file to which all error output produced by tcltest should be written. A file named filename will be [open]ed for writing, and the resulting channel will be set as the value of [errorChannel].

CREATING TEST SUITES WITH TCLTEST

The fundamental element of a test suite is the individual [test] command. We begin with several examples.

[1] Test of a script that returns normally.

```
test example-1.0 {normal return} {
  format %s value
} value
```

[2] Test of a script that requires context setup and cleanup. Note the bracing and indenting style that avoids any need for line continuation.

```
test example-1.1 {test file existence} −setup {
  set file [makeFile {} test]
} −body {
  file exists $file
} −cleanup {
  removeFile test
} −result 1
```


```
test example-1.2 {error return} −body {
  error message
} −returnCodes error −result message
```


```
test example-1.3 {user owns created files} −constraints {
  unix
```
At the next higher layer of organization, several [test] commands are gathered together into a single test file. Test files should have names with the .test extension, because that is the default pattern used by [runAllTests] to find test files. It is a good rule of thumb to have one test file for each source code file of your project. It is good practice to edit the test file and the source code file together, keeping tests synchronized with code changes.

Most of the code in the test file should be the [test] commands. Use constraints to skip tests, rather than conditional evaluation of [test]. That is, do this:

```tcl
[5]
testConstraint X [expr $myRequirement]
test goodConditionalTest {} X {
    # body
} result
and do not do this:

[6]
if $myRequirement {
    test badConditionalTest {} {
        # body
    } result
}
```

Use the −setup and −cleanup options to establish and release all context requirements of the test body. Do not make tests depend on prior tests in the file. Those prior tests might be skipped. If several consecutive tests require the same context, the appropriate setup and cleanup scripts may be stored in variable for passing to each tests −setup and −cleanup options. This is a better solution than performing setup outside of [test] commands, because the setup will only be done if necessary, and any errors during setup will be reported, and not cause the test file to abort.

A test file should be able to be combined with other test files and not interfere with them, even when [configure −singleproc 1] causes all files to be evaluated in a common interpreter. A simple way to achieve this is to have your tests define all their commands and variables in a namespace that is deleted when the test file evaluation is complete. A good namespace to use is a child namespace test of the namespace of the module you are testing.

A test file should also be able to be evaluated directly as a script, not depending on being called by a master [runAllTests]. This means that each test file should process command line arguments to give the tester all the configuration control that tcltest provides.
After all [test]s in a test file, the command [cleanupTests] should be called.

Here is a sketch of a sample test file illustrating those points:

```tcl
package require tcltest 2.2
eval ::tcltest::configure $argv
package require example
namespace eval ::example::test {
    namespace import ::tcltest::*
    testConstraint X [expr {...}]
    variable SETUP {#common setup code}
    variable CLEANUP {#common cleanup code}
    test example-1 {} −setup $SETUP −body {
        # First test
        } −cleanup $CLEANUP −result {...}
    test example-2 {} −constraints X −setup $SETUP −body {
        # Second test; constrained
        } −cleanup $CLEANUP −result {...}
    test example-3 {} {
        # Third test; no context required
        } {...}
    cleanupTests
}
namespace delete ::example::test
```

The next level of organization is a full test suite, made up of several test files. One script is used to control the entire suite. The basic function of this script is to call [runAllTests] after doing any necessary setup. This script is usually named all.tcl because that's the default name used by runAllTests when combining multiple test suites into one testing run.

Here is a sketch of a sample test suite master script:

```tcl
package require Tcl 8.4
package require tcltest 2.2
package require example
::tcltest::configure −testdir
    [file dirname [file normalize [info script]]]
eval ::tcltest::configure $argv ::tcltest::runAllTests
```

COMPATIBILITY

A number of commands and variables in the ::tcltest namespace provided by earlier releases of tcltest have not been documented here. They are no longer part of the supported public interface of tcltest and should not be used in new test suites. However, to continue to support existing test suites written to the older interface specifications, many of those deprecated commands and variables still work as before. For example, in many circumstances, [configure] will be automatically called shortly after [package require tcltest 2.1] succeeds with arguments from the variable ::argv. This is to support test suites that depend on the old behavior that
tcltest was automatically configured from command line arguments. New test files should not depend on this, but should explicitly include

```tcl
eval ::tcltest::configure $::argv
```

to establish a configuration from command line arguments.

**KNOWN ISSUES**

There are two known issues related to nested evaluations of `test`. The first issue relates to the stack level in which test scripts are executed. Tests nested within other tests may be executed at the same stack level as the outermost test. For example, in the following code:

```tcl
test level-1.1 {level 1} {
    -body {
        test level-2.1 {level 2} {
        }
    }
}
```

any script executed in level-2.1 may be executed at the same stack level as the script defined for level-1.1.

In addition, while two `test`s have been run, results will only be reported by `cleanupTests` for tests at the same level as test level-1.1. However, test results for all tests run prior to level-1.1 will be available when test level-2.1 runs. What this means is that if you try to access the test results for test level-2.1, it will may say that 'm' tests have run, 'n' tests have been skipped, 'o' tests have passed and 'p' tests have failed, where 'm', 'n', 'o', and 'p' refer to tests that were run at the same test level as test level-1.1.

Implementation of output and error comparison in the test command depends on usage of `::puts` in your application code. Output is intercepted by redefining the `::puts` command while the defined test script is being run. Errors thrown by C procedures or printed directly from C applications will not be caught by the test command. Therefore, usage of the `−output` and `−errorOutput` options to `test` is useful only for pure Tcl applications that use `::puts` to produce output.

**KEYWORDS**

test, test harness, test suite
tclvars

NAME
tclvars – Variables used by Tcl

DESCRIPTION
env

LOGIN
USER
SYS_FOLDER
APPLE_M_FOLDER
CP_FOLDER
DESK_FOLDER
EXT_FOLDER
PREF_FOLDER
PRINT_MON_FOLDER
SHARED_TRASH_FOLDER
TRASH_FOLDER
START_UP_FOLDER
HOME

errorCode

ARITH code msg
CHILDKILLED pid sigName msg
CHILDSTATUS pid code
CHILDSUSP pid sigName msg
NONE
POSIX errName msg

errorInfo
tcl_library
tcl_patchLevel
tcl_pkgPath
tcl_platform
byteOrder
debug
machine
os
osVersion
platform
threaded
user
wordSize
tcl_precision
tcl_rcFileName
tcl_rcRsrcName
tcl_traceCompile  
tcl_traceExec  
tcl_wordchars  
tcl_nonwordchars  
tcl_version

OTHER GLOBAL VARIABLES
argc  
argv  
argv0  
tcl_interactive  
geometry

SEE ALSO

KEYWORDS

NAME

tclvars – Variables used by Tcl

DESCRIPTION

The following global variables are created and managed automatically by the Tcl library. Except where noted below, these variables should normally be treated as read-only by application-specific code and by users.

env

This variable is maintained by Tcl as an array whose elements are the environment variables for the process. Reading an element will return the value of the corresponding environment variable. Setting an element of the array will modify the corresponding environment variable or create a new one if it doesn't already exist. Unsetting an element of env will remove the corresponding environment variable. Changes to the env array will affect the environment passed to children by commands like exec. If the entire env array is unset then Tcl will stop monitoring env accesses and will not update environment variables.

Under Windows, the environment variables PATH and COMSPEC in any capitalization are converted automatically to upper case. For instance, the PATH variable could be exported by the operating system as ``path", ``Path", ``PaTh", etc., causing otherwise simple Tcl code to have to support many special cases. All other environment variables inherited by Tcl are left unmodified. Setting an env array variable to blank is the same as unsetting it as this is the behavior of the underlying Windows OS. It should be noted that relying on an existing and empty environment variable won't work on windows and is discouraged for cross-platform usage.

On the Macintosh, the environment variable is constructed by Tcl as no global environment variable exists. The environment variables that are created for Tcl include:

LOGIN

This holds the Chooser name of the Macintosh.

USER
This also holds the Chooser name of the Macintosh.

**SYS_FOLDER**
- The path to the system directory.

**APPLE_M_FOLDER**
- The path to the Apple Menu directory.

**CP_FOLDER**
- The path to the control panels directory.

**DESK_FOLDER**
- The path to the desk top directory.

**EXT_FOLDER**
- The path to the system extensions directory.

**PREF_FOLDER**
- The path to the preferences directory.

**PRINT_MON_FOLDER**
- The path to the print monitor directory.

**SHARED_TRASH_FOLDER**
- The path to the network trash directory.

**TRASH_FOLDER**
- The path to the trash directory.

**START_UP_FOLDER**
- The path to the start up directory.

**HOME**
- The path to the application's default directory.

You can also create your own environment variables for the Macintosh. A file named *Tcl Environment Variables* may be placed in the preferences folder in the Mac system folder. Each line of this file should be of the form `VAR_NAME=var_data`.

The last alternative is to place environment variables in a 'STR#' resource named *Tcl Environment Variables* of the application. This is considered a little more ``Mac like'' than a Unix style Environment Variable file. Each entry in the 'STR#' resource has the same format as above. The source code file `tclMacEnv.c` contains the implementation of the env mechanisms. This file contains many #define's that allow customization of the env mechanisms to fit your applications needs.

**errorCode**
After an error has occurred, this variable will be set to hold additional information about the error in a form that is easy to process with programs. **errorCode** consists of a Tcl list with one or more elements. The first element of the list identifies a general class of errors, and determines the format of the rest of the list. The following formats for **errorCode** are used by the Tcl core; individual applications may define additional formats.

**ARITH code msg**
- This format is used when an arithmetic error occurs (e.g. an attempt to divide by zero in the *expr* command). *Code* identifies the precise error and *msg* provides a human-readable description of the error. *Code* will be either DIVZERO (for an attempt to divide by zero), DOMAIN (if an argument is outside the domain of a function, such as acos(−3)),...
IOVERFLOW (for integer overflow), OVERFLOW (for a floating−point overflow), or
UNKNOWN (if the cause of the error cannot be determined).

**CHILDKILLED pid sigName msg**
This format is used when a child process has been killed because of a signal. The second
 element of errorCode will be the process's identifier (in decimal). The third element will be
the symbolic name of the signal that caused the process to terminate; it will be one of the
names from the include file signal.h, such as SIGPIPE. The fourth element will be a short
human−readable message describing the signal, such as ``write on pipe with no readers“ for
SIGPIPE.

**CHILDSTATUS pid code**
This format is used when a child process has exited with a non−zero exit status. The second
 element of errorCode will be the process's identifier (in decimal) and the third element will
be the exit code returned by the process (also in decimal).

**CHILDSUSP pid sigName msg**
This format is used when a child process has been suspended because of a signal. The second
 element of errorCode will be the process's identifier, in decimal. The third element will be
the symbolic name of the signal that caused the process to suspend; this will be one of the
names from the include file signal.h, such as SIGTTIN. The fourth element will be a short
human−readable message describing the signal, such as ``background tty read“ for SIGTTIN.

**NONE**
This format is used for errors where no additional information is available for an error besides
the message returned with the error. In these cases errorCode will consist of a list containing
a single element whose contents are NONE.

**POSIX errName msg**
If the first element of errorCode is POSIX, then the error occurred during a POSIX kernel
call. The second element of the list will contain the symbolic name of the error that occurred,
such as ENOENT; this will be one of the values defined in the include file errno.h. The third
element of the list will be a human−readable message corresponding to errName, such as ``no
such file or directory“ for the ENOENT case.

To set errorCode, applications should use library procedures such as Tcl_SetErrorCode and
Tcl_PosixError, or they may invoke the error command. If one of these methods hasn't been used,
then the Tcl interpreter will reset the variable to NONE after the next error.

**errorInfo**
After an error has occurred, this string will contain one or more lines identifying the Tcl commands
and procedures that were being executed when the most recent error occurred. Its contents take the
form of a stack trace showing the various nested Tcl commands that had been invoked at the time of
the error.

**tcl_library**
This variable holds the name of a directory containing the system library of Tcl scripts, such as those
used for auto−loading. The value of this variable is returned by the info library command. See the
library manual entry for details of the facilities provided by the Tcl script library. Normally each
application or package will have its own application−specific script library in addition to the Tcl
script library; each application should set a global variable with a name like $app_library (where app
is the application's name) to hold the network file name for that application's library directory. The
initial value of tcl_library is set when an interpreter is created by searching several different
directories until one is found that contains an appropriate Tcl startup script. If the `TCL_LIBRARY` environment variable exists, then the directory it names is checked first. If `TCL_LIBRARY` isn't set or doesn't refer to an appropriate directory, then Tcl checks several other directories based on a compiled—in default location, the location of the binary containing the application, and the current working directory.

**tcl_patchLevel**

When an interpreter is created Tcl initializes this variable to hold a string giving the current patch level for Tcl, such as **7.3p2** for Tcl 7.3 with the first two official patches, or **7.4b4** for the fourth beta release of Tcl 7.4. The value of this variable is returned by the `info patchlevel` command.

**tcl_pkgPath**

This variable holds a list of directories indicating where packages are normally installed. It is not used on Windows. It typically contains either one or two entries; if it contains two entries, the first is normally a directory for platform–dependent packages (e.g., shared library binaries) and the second is normally a directory for platform–independent packages (e.g., script files). Typically a package is installed as a subdirectory of one of the entries in `$tcl_pkgPath`. The directories in `$tcl_pkgPath` are included by default in the `auto_path` variable, so they and their immediate subdirectories are automatically searched for packages during `package require` commands. Note: `$tcl_pkgPath` it not intended to be modified by the application. Its value is added to `auto_path` at startup; changes to `$tcl_pkgPath` are not reflected in `auto_path`. If you want Tcl to search additional directories for packages you should add the names of those directories to `auto_path`, not `$tcl_pkgPath`.

**tcl_platform**

This is an associative array whose elements contain information about the platform on which the application is running, such as the name of the operating system, its current release number, and the machine's instruction set. The elements listed below will always be defined, but they may have empty strings as values if Tcl couldn't retrieve any relevant information. In addition, extensions and applications may add additional values to the array. The predefined elements are:

- **byteOrder**
  The native byte order of this machine: either **littleEndian** or **bigEndian**.

- **debug**
  If this variable exists, then the interpreter was compiled with and linked to a debug–enabled C run–time. This variable will only exist on Windows, so extension writers can specify which package to load depending on the C run–time library that is in use. This is not an indication that this core contains symbols.

- **machine**
  The instruction set executed by this machine, such as **intel**, **PPC**, **68k**, or **sun4m**. On UNIX machines, this is the value returned by `uname –m`.

- **os**
  The name of the operating system running on this machine, such as **Windows 95**, **Windows NT**, **MacOS**, or **SunOS**. On UNIX machines, this is the value returned by `uname –s`. On Windows 95 and Windows 98, the value returned will be **Windows 95** to provide better backwards compatibility to Windows 95; to distinguish between the two, check the `osVersion`.

- **osVersion**
  The version number for the operating system running on this machine. On UNIX machines,
this is the value returned by `uname -r`. On Windows 95, the version will be 4.0; on Windows 98, the version will be 4.10.

**platform**

Either `windows`, `macintosh`, or `unix`. This identifies the general operating environment of the machine.

**threaded**

If this variable exists, then the interpreter was compiled with threads enabled.

**user**

This identifies the current user based on the login information available on the platform. This comes from the USER or LOGNAME environment variable on Unix, and the value from GetUser Name on Windows and Macintosh.

**wordSize**

This gives the size of the native−machine word in bytes (strictly, it is same as the result of evaluating `sizeof(long)` in C.)

**tcl_precision**

This variable controls the number of digits to generate when converting floating−point values to strings. It defaults to 12. 17 digits is ``perfect” for IEEE floating−point in that it allows double−precision values to be converted to strings and back to binary with no loss of information. However, using 17 digits prevents any rounding, which produces longer, less intuitive results. For example, `expr 1.4` returns 1.39999999999999999 with `tcl_precision` set to 17, vs. 1.4 if `tcl_precision` is 12.

All interpreters in a process share a single `tcl_precision` value: changing it in one interpreter will affect all other interpreters as well. However, safe interpreters are not allowed to modify the variable.

**tcl_rcFileName**

This variable is used during initialization to indicate the name of a user−specific startup file. If it is set by application−specific initialization, then the Tcl startup code will check for the existence of this file and `source` it if it exists. For example, for `wish` the variable is set to `~/.wishrc` for Unix and `~/wishrc.tcl` for Windows.

**tcl_rcRsrcName**

This variable is only used on Macintosh systems. The variable is used during initialization to indicate the name of a user−specific `TEXT` resource located in the application or extension resource forks. If it is set by application−specific initialization, then the Tcl startup code will check for the existence of this resource and `source` it if it exists. For example, the Macintosh `wish` application has the variable is set to `tclshrc`.

**tcl_traceCompile**

The value of this variable can be set to control how much tracing information is displayed during bytecode compilation. By default, `tcl_traceCompile` is zero and no information is displayed. Setting `tcl_traceCompile` to 1 generates a one−line summary in stdout whenever a procedure or top−level command is compiled. Setting it to 2 generates a detailed listing in stdout of the bytecode instructions emitted during every compilation. This variable is useful in tracking down suspected problems with the Tcl compiler. It is also occasionally useful when converting existing code to use Tcl8.0.

This variable and functionality only exist if `TCL_COMPILE_DEBUG` was defined during Tcl's compilation.
**tcl_traceExec**

The value of this variable can be set to control how much tracing information is displayed during bytecode execution. By default, tcl_traceExec is zero and no information is displayed. Setting tcl_traceExec to 1 generates a one-line trace in stdout on each call to a Tcl procedure. Setting it to 2 generates a line of output whenever any Tcl command is invoked that contains the name of the command and its arguments. Setting it to 3 produces a detailed trace showing the result of executing each bytecode instruction. Note that when tcl_traceExec is 2 or 3, commands such as `set` and `incr` that have been entirely replaced by a sequence of bytecode instructions are not shown. Setting this variable is useful in tracking down suspected problems with the bytecode compiler and interpreter. It is also occasionally useful when converting code to use Tcl8.0.

This variable and functionality only exist if **TCL_COMPILE_DEBUG** was defined during Tcl's compilation.

**tcl_wordchars**

The value of this variable is a regular expression that can be set to control what are considered “word” characters, for instances like selecting a word by double-clicking in text in Tk. It is platform dependent. On Windows, it defaults to `\S`, meaning anything but a Unicode space character. Otherwise it defaults to `\w`, which is any Unicode word character (number, letter, or underscore).

**tcl_nonwordchars**

The value of this variable is a regular expression that can be set to control what are considered “non-word” characters, for instances like selecting a word by double-clicking in text in Tk. It is platform dependent. On Windows, it defaults to `\s`, meaning any Unicode space character. Otherwise it defaults to `\W`, which is anything but a Unicode word character (number, letter, or underscore).

**tcl_version**

When an interpreter is created Tcl initializes this variable to hold the version number for this version of Tcl in the form `x.y`. Changes to `x` represent major changes with probable incompatibilities and changes to `y` represent small enhancements and bug fixes that retain backward compatibility. The value of this variable is returned by the `info tclversion` command.

### OTHER GLOBAL VARIABLES

The following variables are only guaranteed to exist in **tclsh** and **wish** executables; the Tcl library does not define them itself but many Tcl environments do.

**argc**

The number of arguments to **tclsh** or **wish**.

**argv**

Tcl list of arguments to **tclsh** or **wish**.

**argv0**

The script that **tclsh** or **wish** started executing (if it was specified) or otherwise the name by which **tclsh** or **wish** was invoked.

**tcl_interactive**

Contains 1 if **tclsh** or **wish** is running interactively (no script was specified and standard input is a terminal–like device), 0 otherwise.
The `wish` executably additionally specifies the following global variable:

**geometry**

If set, contains the user-supplied geometry specification to use for the main Tk window.

**SEE ALSO**

`eval`, `tclsh`, `wish`

**KEYWORDS**

`arithmetic`, `bytecode`, `compiler`, `error`, `environment`, `POSIX`, `precision`, `subprocess`, `variables`
tell

NAME
tell – Return current access position for an open channel

SYNOPSIS
tell channelId

DESCRIPTION
Returns an integer string giving the current access position in channelId. This value returned is a byte offset that can be passed to seek in order to set the channel to a particular position. Note that this value is in terms of bytes, not characters like read. The value returned is −1 for channels that do not support seeking.

ChannelId must be an identifier for an open channel such as a Tcl standard channel (stdin, stdout, or stderr), the return value from an invocation of open or socket, or the result of a channel creation command provided by a Tcl extension.

EXAMPLE
Read a line from a file channel only if it starts with foobar:

```
# Save the offset in case we need to undo the read...
set offset [tell $chan]
if {[read $chan 6] eq "foobar"} {
  gets $chan line
} else {
  set line {}
  # Undo the read...
  seek $chan $offset
}
```

SEE ALSO
file, open, close, gets, seek, Tcl_StandardChannels

KEYWORDS
access position, channel, seeking

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
time

NAME

time – Time the execution of a script

SYNOPSIS

time script ?count?

DESCRIPTION

This command will call the Tcl interpreter count times to evaluate script (or once if count isn't specified). It will then return a string of the form

503 microseconds per iteration

which indicates the average amount of time required per iteration, in microseconds. Time is measured in elapsed time, not CPU time.

EXAMPLE

Estimate how long it takes for a simple Tcl for loop to count to a thousand:

time {
    for {set i 0} {$i<1000} {incr i} {
        # empty body
    }
}

SEE ALSO

clock

KEYWORDS

script, time
NAME

trace – Monitor variable accesses, command usages and command executions

SYNOPSIS

trace option ?arg arg ...?

DESCRIPTION

trace add type name ops ?args?

trace add command name ops command
 rename
 delete

trace add execution name ops command
 enter
 leave
 enterstep
 leavestep

trace add variable name ops command
 array
 read
 write
 unset

trace remove type name opList command

trace remove command name opList command
 trace remove execution name opList command
 trace remove variable name opList command

trace info type name

trace info command name
 trace info execution name
 trace info variable name

trace variable name ops command

trace vdelete name ops command

trace vinfo name

EXAMPLES

SEE ALSO

KEYWORDS

NAME

trace – Monitor variable accesses, command usages and command executions
SYNOPSIS

trace option ?arg arg ...?

DESCRIPTION

This command causes Tcl commands to be executed whenever certain operations are invoked. The legal
option's (which may be abbreviated) are:

trace add type name ops ?args?

Where type is command, execution, or variable.

trace add command name ops command

Arrange for command to be executed whenever command name is modified in one of the
ways given by the list ops. Name will be resolved using the usual namespace resolution rules
used by procedures. If the command does not exist, an error will be thrown.

Ops indicates which operations are of interest, and is a list of one or more of the following items:

rename

Invoke command whenever the command is renamed. Note that renaming to the
empty string is considered deletion, and will not be traced with 'rename'.

delete

Invoke command when the command is deleted. Commands can be deleted explicitly
by using the rename command to rename the command to an empty string.
Commands are also deleted when the interpreter is deleted, but traces will not be
invoked because there is no interpreter in which to execute them.

When the trace triggers, depending on the operations being traced, a number of arguments are
appended to command so that the actual command is as follows:

command oldName newName op

OldName and newName give the traced command's current (old) name, and the name to
which it is being renamed (the empty string if this is a 'delete' operation). Op indicates what
operation is being performed on the command, and is one of rename or delete as defined
above. The trace operation cannot be used to stop a command from being deleted. Tcl will
always remove the command once the trace is complete. Recursive renaming or deleting will
not cause further traces of the same type to be evaluated, so a delete trace which itself deletes
the command, or a rename trace which itself renames the command will not cause further
trace evaluations to occur. Both oldName and newName are fully qualified with any
namespace(s) in which they appear.

trace add execution name ops command

Arrange for command to be executed whenever command name is executed, with traces
occurring at the points indicated by the list ops. Name will be resolved using the usual
namespace resolution rules used by procedures. If the command does not exist, an error will be thrown.

*Ops* indicates which operations are of interest, and is a list of one or more of the following items:

**enter**
Invoke *command* whenever the command *name* is executed, just before the actual execution takes place.

**leave**
Invoke *command* whenever the command *name* is executed, just after the actual execution takes place.

**enterstep**
Invoke *command* for every Tcl command which is executed inside the procedure *name*, just before the actual execution takes place. For example if we have 'proc foo {} { puts "hello" }', then an enterstep trace would be invoked just before `puts "hello"` is executed. Setting an enterstep trace on a *command* will not result in an error and is simply ignored.

**leavestep**
Invoke *command* for every Tcl command which is executed inside the procedure *name*, just after the actual execution takes place. Setting a leavestep trace on a *command* will not result in an error and is simply ignored.

When the trace triggers, depending on the operations being traced, a number of arguments are appended to *command* so that the actual command is as follows:

For **enter** and **enterstep** operations:

```
command command-string op
```

*Command-string* gives the complete current command being executed (the traced command for a **enter** operation, an arbitrary command for a **enterstep** operation), including all arguments in their fully expanded form. *Op* indicates what operation is being performed on the command execution, and is one of **enter** or **enterstep** as defined above. The trace operation can be used to stop the command from executing, by deleting the command in question. Of course when the command is subsequently executed, an 'invalid command' error will occur.

For **leave** and **leavestep** operations:

```
command command-string code result op
```

*Command-string* gives the complete current command being executed (the traced command for a **enter** operation, an arbitrary command for a **enterstep** operation), including all arguments in their fully expanded form. *Code* gives the result code of that execution, and *result* the result string. *Op* indicates what operation is being performed on the command execution, and is one of **leave** or **leavestep** as defined above. Note that the creation of many
**enterstep** or **leavestep** traces can lead to unintuitive results, since the invoked commands from one trace can themselves lead to further command invocations for other traces.

*Command* executes in the same context as the code that invoked the traced operation: thus the *command*, if invoked from a procedure, will have access to the same local variables as code in the procedure. This context may be different than the context in which the trace was created. If *command* invokes a procedure (which it normally does) then the procedure will have to use upvar or uplevel commands if it wishes to access the local variables of the code which invoked the trace operation.

While *command* is executing during an execution trace, traces on *name* are temporarily disabled. This allows the *command* to execute *name* in its body without invoking any other traces again. If an error occurs while executing the *command* body, then the command *name* as a whole will return that same error.

When multiple traces are set on *name*, then for **enter** and **enterstep** operations, the traced commands are invoked in the reverse order of how the traces were originally created; and for **leave** and **leavestep** operations, the traced commands are invoked in the original order of creation.

The behavior of execution traces is currently undefined for a command *name* imported into another namespace.

**trace add variable** *name* *ops* *command*

Arrange for *command* to be executed whenever variable *name* is accessed in one of the ways given by the list *ops*. *Name* may refer to a normal variable, an element of an array, or to an array as a whole (i.e. *name* may be just the name of an array, with no parenthesized index). If *name* refers to a whole array, then *command* is invoked whenever any element of the array is manipulated. If the variable does not exist, it will be created but will not be given a value, so it will be visible to *namespace which* queries, but not to info exists queries.

*Ops* indicates which operations are of interest, and is a list of one or more of the following items:

- **array**
  Invoke *command* whenever the variable is accessed or modified via the *array* command, provided that *name* is not a scalar variable at the time that the *array* command is invoked. If *name* is a scalar variable, the access via the *array* command will not trigger the trace.

- **read**
  Invoke *command* whenever the variable is read.

- **write**
  Invoke *command* whenever the variable is written.

- **unset**
  Invoke *command* whenever the variable is unset. Variables can be unset explicitly with the *unset* command, or implicitly when procedures return (all of their local...
variables are unset). Variables are also unset when interpreters are deleted, but traces will not be invoked because there is no interpreter in which to execute them.

When the trace triggers, three arguments are appended to command so that the actual command is as follows:

```
command name1 name2 op
```

Name1 and name2 give the name(s) for the variable being accessed: if the variable is a scalar then name1 gives the variable's name and name2 is an empty string; if the variable is an array element then name1 gives the name of the array and name2 gives the index into the array; if an entire array is being deleted and the trace was registered on the overall array, rather than a single element, then name1 gives the array name and name2 is an empty string. Name1 and name2 are not necessarily the same as the name used in the trace variable command: the upvar command allows a procedure to reference a variable under a different name. Op indicates what operation is being performed on the variable, and is one of read, write, or unset as defined above.

Command executes in the same context as the code that invoked the traced operation: if the variable was accessed as part of a Tcl procedure, then command will have access to the same local variables as code in the procedure. This context may be different than the context in which the trace was created. If command invokes a procedure (which it normally does) then the procedure will have to use upvar or uplevel if it wishes to access the traced variable. Note also that name1 may not necessarily be the same as the name used to set the trace on the variable; differences can occur if the access is made through a variable defined with the upvar command.

For read and write traces, command can modify the variable to affect the result of the traced operation. If command modifies the value of a variable during a read or write trace, then the new value will be returned as the result of the traced operation. The return value from command is ignored except that if it returns an error of any sort then the traced operation also returns an error with the same error message returned by the trace command (this mechanism can be used to implement read–only variables, for example). For write traces, command is invoked after the variable's value has been changed; it can write a new value into the variable to override the original value specified in the write operation. To implement read–only variables, command will have to restore the old value of the variable.

While command is executing during a read or write trace, traces on the variable are temporarily disabled. This means that reads and writes invoked by command will occur directly, without invoking command (or any other traces) again. However, if command unsets the variable then unset traces will be invoked.

When an unset trace is invoked, the variable has already been deleted: it will appear to be undefined with no traces. If an unset occurs because of a procedure return, then the trace will be invoked in the variable context of the procedure being returned to: the stack frame of the returning procedure will no longer exist. Traces are not disabled during unset traces, so if an unset trace command creates a new trace and accesses the variable, the trace will be invoked.
Any errors in unset traces are ignored.

If there are multiple traces on a variable they are invoked in order of creation, most−recent first. If one trace returns an error, then no further traces are invoked for the variable. If an array element has a trace set, and there is also a trace set on the array as a whole, the trace on the overall array is invoked before the one on the element.

Once created, the trace remains in effect either until the trace is removed with the trace remove variable command described below, until the variable is unset, or until the interpreter is deleted. Unsetting an element of array will remove any traces on that element, but will not remove traces on the overall array.

This command returns an empty string.

\texttt{trace remove type name opList command}

Where \texttt{type} is either \texttt{command, execution} or \texttt{variable}.

\texttt{trace remove command name opList command}

If there is a trace set on command \texttt{name} with the operations and command given by \texttt{opList} and \texttt{command}, then the trace is removed, so that \texttt{command} will never again be invoked.

Returns an empty string. If \texttt{name} doesn't exist, the command will throw an error.

\texttt{trace remove execution name opList command}

If there is a trace set on command \texttt{name} with the operations and command given by \texttt{opList} and \texttt{command}, then the trace is removed, so that \texttt{command} will never again be invoked.

Returns an empty string. If \texttt{name} doesn't exist, the command will throw an error.

\texttt{trace remove variable name opList command}

If there is a trace set on variable \texttt{name} with the operations and command given by \texttt{opList} and \texttt{command}, then the trace is removed, so that \texttt{command} will never again be invoked. Returns an empty string.

\texttt{trace info type name}

Where \texttt{type} is either \texttt{command, execution} or \texttt{variable}.

\texttt{trace info command name}

Returns a list containing one element for each trace currently set on command \texttt{name}. Each element of the list is itself a list containing two elements, which are the \texttt{opList} and \texttt{command} associated with the trace. If \texttt{name} doesn't have any traces set, then the result of the command will be an empty string. If \texttt{name} doesn't exist, the command will throw an error.

\texttt{trace info execution name}

Returns a list containing one element for each trace currently set on command \texttt{name}. Each element of the list is itself a list containing two elements, which are the \texttt{opList} and \texttt{command} associated with the trace. If \texttt{name} doesn't have any traces set, then the result of the command will be an empty string. If \texttt{name} doesn't exist, the command will throw an error.

\texttt{trace info variable name}

Returns a list containing one element for each trace currently set on variable \texttt{name}. Each element of the list is itself a list containing two elements, which are the \texttt{opList} and \texttt{command} associated with the trace. If \texttt{name} doesn't exist or doesn't have any traces set, then the result of
the command will be an empty string.

For backwards compatibility, three other subcommands are available:

- **trace variable name ops command**
  - This is equivalent to `trace add variable name ops command`.
- **trace vdelete name ops command**
  - This is equivalent to `trace remove variable name ops command`.
- **trace vinfo name**
  - This is equivalent to `trace info variable name`.

These subcommands are deprecated and will likely be removed in a future version of Tcl. They use an older syntax in which `array`, `read`, `write`, `unset` are replaced by `a`, `r`, `w` and `u` respectively, and the `ops` argument is not a list, but simply a string concatenation of the operations, such as `rwua`.

**EXAMPLES**

Print a message whenever either of the global variables `foo` and `bar` are updated, even if they have a different local name at the time (which can be done with the `upvar` command):

```tcl
proc tracer {varname args} {
    upvar #0 $varname var
    puts "$varname was updated to be "$var"
}
trace add variable foo write "tracer foo"
trace add variable bar write "tracer bar"
```

Ensure that the global variable `foobar` always contains the product of the global variables `foo` and `bar`:

```tcl
proc doMult args {
    global foo bar foobar
    set foobar [expr {$foo * $bar}]
}
trace add variable foo write doMult
trace add variable bar write doMult
```

**SEE ALSO**

- `set`, `unset`

**KEYWORDS**

- `read`, `command`, `rename`, `variable`, `write`, `trace`, `unset`
unknown

NAME

unknown – Handle attempts to use non-existent commands

SYNOPSIS

unknown cmdName ?arg arg ...?

DESCRIPTION

This command is invoked by the Tcl interpreter whenever a script tries to invoke a command that doesn't exist. The default implementation of unknown is a library procedure defined when Tcl initializes an interpreter. You can override the default unknown to change its functionality. Note that there is no default implementation of unknown in a safe interpreter.

If the Tcl interpreter encounters a command name for which there is not a defined command, then Tcl checks for the existence of a command named unknown. If there is no such command, then the interpreter returns an error. If the unknown command exists, then it is invoked with arguments consisting of the fully-substituted name and arguments for the original non-existent command. The unknown command typically does things like searching through library directories for a command procedure with the name cmdName, or expanding abbreviated command names to full-length, or automatically executing unknown commands as sub-processes. In some cases (such as expanding abbreviations) unknown will change the original command slightly and then (re-)execute it. The result of the unknown command is used as the result for the original non-existent command.

The default implementation of unknown behaves as follows. It first calls the auto_load library procedure to load the command. If this succeeds, then it executes the original command with its original arguments. If the auto-load fails then unknown calls auto_execok to see if there is an executable file by the name cmd. If so, it invokes the Tcl exec command with cmd and all the args as arguments. If cmd can't be auto-executed, unknown checks to see if the command was invoked at top-level and outside of any script. If so, then unknown takes two additional steps. First, it sees if cmd has one of the following three forms: !!, !event, or ^old^new?!??. If so, then unknown carries out history substitution in the same way that csh would for these constructs. Finally, unknown checks to see if cmd is a unique abbreviation for an existing Tcl command. If so, it expands the command name and executes the command with the original arguments. If none of the above efforts has been able to execute the command, unknown generates an error return. If the global variable auto_noload is defined, then the auto-load step is skipped. If the global variable auto_noexec is defined then the auto-exec step is skipped. Under normal circumstances the return value from unknown is the return value from the command that was eventually executed.
EXAMPLE

Arrange for the **unknown** command to have its standard behavior except for first logging the fact that a command was not found:

```tcl
# Save the original one so we can chain to it
rename unknown _original_unknown

# Provide our own implementation
proc unknown args {
    puts stderr "WARNING: unknown command: $args"
    uplevel 1 [list _original_unknown {expand}$args]
}
```

SEE ALSO

`info`, `proc`, `interp`, `library`

KEYWORDS

`error`, `non-existent command`

---

*Copyright ' 1993 The Regents of the University of California.*  
*Copyright ' 1994-1996 Sun Microsystems, Inc.*  
*Copyright ' 1995-1997 Roger E. Critchlow Jr.*
unset

NAME

unset – Delete variables

SYNOPSIS

unset ?–nocomplain? ?--? ?name name name ...?

DESCRIPTION

This command removes one or more variables. Each name is a variable name, specified in any of the ways acceptable to the set command. If a name refers to an element of an array then that element is removed without affecting the rest of the array. If a name consists of an array name with no parenthesized index, then the entire array is deleted. The unset command returns an empty string as result. If –nocomplain is specified as the first argument, any possible errors are suppressed. The option may not be abbreviated, in order to disambiguate it from possible variable names. The option -- indicates the end of the options, and should be used if you wish to remove a variable with the same name as any of the options. If an error occurs, any variables after the named one causing the error not deleted. An error can occur when the named variable doesn't exist, or the name refers to an array element but the variable is a scalar, or the name refers to a variable in a non-existent namespace.

EXAMPLE

Create an array containing a mapping from some numbers to their squares and remove the array elements for non-prime numbers:

```bash
array set squares {
    1 1 6 36
    2 4 7 49
    3 9 8 64
    4 16 9 81
    5 25 10 100
}
puts "The squares are:"
parray squares
unset squares(1) squares(4) squares(6)
unset squares(8) squares(9) squares(10)
puts "The prime squares are:"
parray squares
```
SEE ALSO

*set, trace, upvar*

KEYWORDS

*remove, variable*
update

NAME

update – Process pending events and idle callbacks

SYNOPSIS

update ?idletasks?

DESCRIPTION

This command is used to bring the application ``up to date'' by entering the event loop repeatedly until all pending events (including idle callbacks) have been processed.

If the idletasks keyword is specified as an argument to the command, then no new events or errors are processed; only idle callbacks are invoked. This causes operations that are normally deferred, such as display updates and window layout calculations, to be performed immediately.

The update idletasks command is useful in scripts where changes have been made to the application's state and you want those changes to appear on the display immediately, rather than waiting for the script to complete. Most display updates are performed as idle callbacks, so update idletasks will cause them to run. However, there are some kinds of updates that only happen in response to events, such as those triggered by window size changes; these updates will not occur in update idletasks.

The update command with no options is useful in scripts where you are performing a long-running computation but you still want the application to respond to events such as user interactions; if you occasionally call update then user input will be processed during the next call to update.

EXAMPLE

Run computations for about a second and then finish:

set x 1000
set done 0
after 1000 set done 1
while {!$done} {
    # A very silly example!
    set x [expr {log($x) ** 2.8}]

    # Test to see if our time-limit has been hit. This would
    # also give a chance for serving network sockets and, if
    # the Tk package is loaded, updating a user interface.
    update
}

update 346
SEE ALSO

*after*, *berror*

KEYWORDS

*event*, *flush*, *handler*, *idle*, *update*
**uplevel**

**NAME**

uplevel – Execute a script in a different stack frame

**SYNOPSIS**

```
uplevel ?level? arg ?arg ...?
```

**DESCRIPTION**

All of the `arg` arguments are concatenated as if they had been passed to `concat`; the result is then evaluated in the variable context indicated by `level`. **Uplevel** returns the result of that evaluation.

If `level` is an integer then it gives a distance (up the procedure calling stack) to move before executing the command. If `level` consists of `#` followed by a number then the number gives an absolute level number. If `level` is omitted then it defaults to `1`. Level cannot be defaulted if the first `command` argument starts with a digit or `#`.

For example, suppose that procedure `a` was invoked from top-level, and that it called `b`, and that `b` called `c`. Suppose that `c` invokes the `uplevel` command. If `level` is `1` or `#2` or omitted, then the command will be executed in the variable context of `b`. If `level` is `2` or `#1` then the command will be executed in the variable context of `a`. If `level` is `3` or `#0` then the command will be executed at top-level (only global variables will be visible).

The `uplevel` command causes the invoking procedure to disappear from the procedure calling stack while the command is being executed. In the above example, suppose `c` invokes the command

```
uplevel 1 (set x 43; d)
```

where `d` is another Tcl procedure. The `set` command will modify the variable `x` in `b`'s context, and `d` will execute at level 3, as if called from `b`. If it in turn executes the command

```
uplevel (set x 42)
```

then the `set` command will modify the same variable `x` in `b`'s context: the procedure `c` does not appear to be on the call stack when `d` is executing. The command ```info level''` may be used to obtain the level of the current procedure.

**Uplevel** makes it possible to implement new control constructs as Tcl procedures (for example, **uplevel** could be used to implement the `while` construct as a Tcl procedure).

**namespace eval** is another way (besides procedure calls) that the Tcl naming context can change. It adds a call frame to the stack to represent the namespace context. This means each **namespace eval** command counts
as another call level for uplevel and upvar commands. For example, info level 1 will return a list describing a command that is either the outermost procedure call or the outermost namespace eval command. Also, uplevel #0 evaluates a script at top−level in the outermost namespace (the global namespace).

EXAMPLE

As stated above, the uplevel command is useful for creating new control constructs. This example shows how (without error handling) it can be used to create a do command that is the counterpart of while except for always performing the test after running the loop body:

```tcl
proc do {body while condition} {
    if {${while ne "while"}} {
        error "required word missing"
    }
    set conditionCmd [list expr $condition]
    while {1} {
        uplevel 1 $body
        if {![uplevel 1 $conditionCmd]} {
            break
        }
    }
}
```

SEE ALSO

namespace, upvar

KEYWORDS

context, level, namespace, stack frame, variables

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1997 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
upvar

NAME

upvar – Create link to variable in a different stack frame

SYNOPSIS

upvar ?level? otherVar myVar ?otherVar myVar ...?

DESCRIPTION

This command arranges for one or more local variables in the current procedure to refer to variables in an enclosing procedure call or to global variables. Level may have any of the forms permitted for the uplevel command, and may be omitted if the first letter of the first otherVar isn't # or a digit (it defaults to 1). For each otherVar argument, upvar makes the variable by that name in the procedure frame given by level (or at global level, if level is #0) accessible in the current procedure by the name given in the corresponding myVar argument. The variable named by otherVar need not exist at the time of the call; it will be created the first time myVar is referenced, just like an ordinary variable. There must not exist a variable by the name myVar at the time upvar is invoked. MyVar is always treated as the name of a variable, not an array element. Even if the name looks like an array element, such as a(b), a regular variable is created. OtherVar may refer to a scalar variable, an array, or an array element. Upvar returns an empty string.

The upvar command simplifies the implementation of call–by–name procedure calling and also makes it easier to build new control constructs as Tcl procedures. For example, consider the following procedure:

```
proc add2 name {
    upvar $name x
    set x [expr $x+2]
}
```

add2 is invoked with an argument giving the name of a variable, and it adds two to the value of that variable. Although add2 could have been implemented using uplevel instead of upvar, upvar makes it simpler for add2 to access the variable in the caller's procedure frame.

namespace eval is another way (besides procedure calls) that the Tcl naming context can change. It adds a call frame to the stack to represent the namespace context. This means each namespace eval command counts as another call level for uplevel and upvar commands. For example, info level 1 will return a list describing a command that is either the outermost procedure call or the outermost namespace eval command. Also, uplevel #0 evaluates a script at top–level in the outermost namespace (the global namespace).

If an upvar variable is unset (e.g. x in add2 above), the unset operation affects the variable it is linked to, not the upvar variable. There is no way to unset an upvar variable except by exiting the procedure in which it is defined. However, it is possible to retarget an upvar variable by executing another upvar command.
TRACES AND UPVAR

Upvar interacts with traces in a straightforward but possibly unexpected manner. If a variable trace is defined on `otherVar`, that trace will be triggered by actions involving `myVar`. However, the trace procedure will be passed the name of `myVar`, rather than the name of `otherVar`. Thus, the output of the following code will be `localVar` rather than `originalVar`:

```tcl
proc traceproc { name index op } {
    puts $name
}
proc setByUpvar { name value } {
    upvar $name localVar
    set localVar $value
}
set originalVar 1
trace variable originalVar w traceproc
setByUpvar originalVar 2
```

If `otherVar` refers to an element of an array, then variable traces set for the entire array will not be invoked when `myVar` is accessed (but traces on the particular element will still be invoked). In particular, if the array is `env`, then changes made to `myVar` will not be passed to subprocesses correctly.

EXAMPLE

A `decr` command that works like `incr` except it subtracts the value from the variable instead of adding it:

```tcl
proc decr {varName {decrement 1}} {
    upvar 1 $varName var
    incr var [expr {-$decrement}]
}
```

SEE ALSO

`global`, `namespace`, `uplevel`, `variable`

KEYWORDS

`context`, `frame`, `global`, `level`, `namespace`, `procedure`, `variable`

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1997 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
variable

NAME

variable – create and initialize a namespace variable

SYNOPSIS

variable ?name value...? name ?value?

DESCRIPTION

This command is normally used within a `namespace eval` command to create one or more variables within a namespace. Each variable `name` is initialized with `value`. The `value` for the last variable is optional.

If a variable `name` does not exist, it is created. In this case, if `value` is specified, it is assigned to the newly created variable. If no `value` is specified, the new variable is left undefined. If the variable already exists, it is set to `value` if `value` is specified or left unchanged if no `value` is given. Normally, `name` is unqualified (does not include the names of any containing namespaces), and the variable is created in the current namespace. If `name` includes any namespace qualifiers, the variable is created in the specified namespace. If the variable is not defined, it will be visible to the `namespace which` command, but not to the `info exists` command.

If the `variable` command is executed inside a Tcl procedure, it creates local variables linked to the corresponding namespace variables (and therefore these variables are listed by `info locals`.) In this way the `variable` command resembles the `global` command, although the `global` command only links to variables in the global namespace. If any `values` are given, they are used to modify the values of the associated namespace variables. If a namespace variable does not exist, it is created and optionally initialized.

A `name` argument cannot reference an element within an array. Instead, `name` should reference the entire array, and the initialization `value` should be left off. After the variable has been declared, elements within the array can be set using ordinary `set` or `array` commands.

EXAMPLES

Create a variable in a namespace:

```tcl
namespace eval foo {
    variable bar 12345
}
```

Create an array in a namespace:

```tcl
namespace eval someNS {
    variable someAry
    array set someAry {
```
namespace eval foo {
    proc spong {} {
        # Variable in this namespace
        variable bar
        puts "bar is $bar"

        # Variable in another namespace
        variable ::someNS::someAry
        parray someAry
    }
}

SEE ALSO

global namespace upvar

KEYWORDS

global namespace procedure variable

Copyright ' 1993-1997 Bell Labs Innovations for Lucent Technologies
Copyright ' 1997 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
vwait

NAME

vwait – Process events until a variable is written

SYNOPSIS

vwait varName

DESCRIPTION

This command enters the Tcl event loop to process events, blocking the application if no events are ready. It continues processing events until some event handler sets the value of variable varName. Once varName has been set, the vwait command will return as soon as the event handler that modified varName completes. varName must globally scoped (either with a call to global for the varName, or with the full namespace path specification).

In some cases the vwait command may not return immediately after varName is set. This can happen if the event handler that sets varName does not complete immediately. For example, if an event handler sets varName and then itself calls vwait to wait for a different variable, then it may not return for a long time. During this time the top–level vwait is blocked waiting for the event handler to complete, so it cannot return either.

EXAMPLES

Run the event–loop continually until some event calls exit. (You can use any variable not mentioned elsewhere, but the name forever reminds you at a glance of the intent.)

vwait forever

Wait five seconds for a connection to a server socket, otherwise close the socket and continue running the script:

# Initialise the state
after 5000 set state timeout
set server [socket –server accept 12345]
proc accept {args} {
    global state connectionInfo
    set state accepted
    set connectionInfo $args
}

# Wait for something to happen
vwait state

# Clean up events that could have happened
close $server
after cancel set state timeout

# Do something based on how the vwait finished...
switch $state {
    timeout {
        puts "no connection on port 12345"
    }
    accepted {
        puts "connection: $connectionInfo"
        puts [lindex $connectionInfo 0] "Hello there!"
    }
}

SEE ALSO

global, update

KEYWORDS

event, variable, wait

Copyright ' 1995-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
while

NAME

while – Execute script repeatedly as long as a condition is met

SYNOPSIS

```tcl
while test body
```

DESCRIPTION

The `while` command evaluates `test` as an expression (in the same way that `expr` evaluates its argument). The value of the expression must be a proper boolean value; if it is a true value then `body` is executed by passing it to the Tcl interpreter. Once `body` has been executed then `test` is evaluated again, and the process repeats until eventually `test` evaluates to a false boolean value. `Continue` commands may be executed inside `body` to terminate the current iteration of the loop, and `break` commands may be executed inside `body` to cause immediate termination of the `while` command. The `while` command always returns an empty string.

Note: `test` should almost always be enclosed in braces. If not, variable substitutions will be made before the `while` command starts executing, which means that variable changes made by the loop body will not be considered in the expression. This is likely to result in an infinite loop. If `test` is enclosed in braces, variable substitutions are delayed until the expression is evaluated (before each loop iteration), so changes in the variables will be visible. For an example, try the following script with and without the braces around `$x<10`:

```tcl
set x 0
while {($x<10)} {
    puts "x is $x"
    incr x
}
```

EXAMPLE

Read lines from a channel until we get to the end of the stream, and print them out with a line-number prepended:

```tcl
set lineCount 0
while {([gets $chan line] >= 0)} {
    puts "[incr lineCount]: $line"
}
```

SEE ALSO

`break`, `continue`, `for`, `foreach`
<table>
<thead>
<tr>
<th>Tk Commands</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bell</td>
<td>entry</td>
</tr>
<tr>
<td>bind</td>
<td>event</td>
</tr>
<tr>
<td>bindtags</td>
<td>focus</td>
</tr>
<tr>
<td>bitmap</td>
<td>font</td>
</tr>
<tr>
<td>button</td>
<td>frame</td>
</tr>
<tr>
<td>canvas</td>
<td>grab</td>
</tr>
<tr>
<td>checkbutton</td>
<td>grid</td>
</tr>
<tr>
<td>clipboard</td>
<td>image</td>
</tr>
<tr>
<td>colors</td>
<td>keysyms</td>
</tr>
<tr>
<td>console</td>
<td>label</td>
</tr>
<tr>
<td>cursors</td>
<td>labelframe</td>
</tr>
<tr>
<td>destroy</td>
<td>listbox</td>
</tr>
</tbody>
</table>

*Copyright ' 1990-1994 The Regents of the University of California.*

*Copyright ' 1994 The Australian National University*

*Copyright ' 1994-1997 Sun Microsystems, Inc.*

*Copyright ' 1997-2000 Scriptics Corporation.*

*Copyright ' 1998-2000 Ajuba Solutions.*

*Copyright ' 2000 Jeffrey Hobbs.*

*Copyright ' 2001 Donal K. Fellows*

*Copyright ' 2003 ActiveState Corporation.*

*Copyright ' 1995-1997 Roger E. Critchlow Jr.*
loadTk

NAME

loadTk – Load Tk into a safe interpreter.

SYNOPSIS

::safe::loadTk slave ?–use windowId? ?–display displayName?

Safe Tk is based on Safe Tcl, which provides a mechanism that allows restricted and mediated access to auto-loading and packages for safe interpreters. Safe Tk adds the ability to configure the interpreter for safe Tk operations and load Tk into safe interpreters.

DESCRIPTION

The ::safe::loadTk command initializes the required data structures in the named safe interpreter and then loads Tk into it. The interpreter must have been created with ::safe::interpCreate or have been initialized with ::safe::interpInit. The command returns the name of the safe interpreter. If –use is specified, the window identified by the specified system dependent identifier windowId is used to contain the ``.`" window of the safe interpreter; it can be any valid id, eventually referencing a window belonging to another application. As a convenience, if the window you plan to use is a Tk Window of the application you can use the window name (e.g. .x.y) instead of its window Id ([winfo id .x.y]). When –use is not specified, a new toplevel window is created for the ``.`" window of the safe interpreter. On X11 if you want the embedded window to use another display than the default one, specify it with –display. See the SECURITY ISSUES section below for implementation details.

SECURITY ISSUES

Please read the safe manual page for Tcl to learn about the basic security considerations for Safe Tcl.

::safe::loadTk adds the value of tk_library taken from the master interpreter to the virtual access path of the safe interpreter so that auto-loading will work in the safe interpreter.

Tk initialization is now safe with respect to not trusting the slave's state for startup. ::safe::loadTk registers the slave's name so when the Tk initialization (Tk_SafeInit) is called and in turn calls the master's ::safe::InitTk it will return the desired argv equivalent (–use windowId, correct –display, etc.)

When –use is not used, the new toplevel created is specially decorated so the user is always aware that the user interface presented comes from a potentially unsafe code and can easily delete the corresponding interpreter.

On X11, conflicting –use and –display are likely to generate a fatal X error.
SEE ALSO

safe, interp, library, load, package, source, unknown

KEYWORDS

alias, auto-loading, auto_mkindex, load, master interpreter, safe interpreter, slave interpreter, source

Copyright ' 1995-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
bell

NAME

bell – Ring a display's bell

SYNOPSIS

bell ?–displayof window? ?–nice?

DESCRIPTION

This command rings the bell on the display for window and returns an empty string. If the –displayof option is omitted, the display of the application's main window is used by default. The command uses the current bell–related settings for the display, which may be modified with programs such as xset.

If –nice is not specified, this command also resets the screen saver for the screen. Some screen savers will ignore this, but others will reset so that the screen becomes visible again.

KEYWORDS

beep, bell, ring

Copyright ' 1994 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 2000 Ajuba Solutions.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

bind – Arrange for X events to invoke Tcl scripts

SYNOPSIS

bind tag ?sequence? ?+??script?

INTRODUCTION

EVENT PATTERNS

MODIFIERS

EVENT TYPES

Activate, Deactivate
MouseWheel
KeyPress, KeyRelease
ButtonPress, ButtonRelease, Motion
Configure
Map, Unmap
Visibility
Expose
Destroy
FocusIn, FocusOut
Enter, Leave
Property
Colormap
MapRequest, CirculateRequest, ResizeRequest, ConfigureRequest, Create
Gravity, Reparent, Circulate

EVENT DETAILS

BINDING SCRIPTS AND SUBSTITUTIONS

%%
%#
%a
%b
%c
%d
%f
%h
%i
%k
%m
%o
%p
%s
%t
%w
NAME

bind – Arrange for X events to invoke Tcl scripts

SYNOPSIS

bind tag ?sequence? ?+??script?

INTRODUCTION

The bind command associates Tcl scripts with X events. If all three arguments are specified, bind will arrange for script (a Tcl script) to be evaluated whenever the event(s) given by sequence occur in the window(s) identified by tag. If script is prefixed with a `\`+"", then it is appended to any existing binding for sequence; otherwise script replaces any existing binding. If script is an empty string then the current binding for sequence is destroyed, leaving sequence unbound. In all of the cases where a script argument is provided, bind returns an empty string.

If sequence is specified without a script, then the script currently bound to sequence is returned, or an empty string is returned if there is no binding for sequence. If neither sequence nor script is specified, then the return value is a list whose elements are all the sequences for which there exist bindings for tag.

The tag argument determines which window(s) the binding applies to. If tag begins with a dot, as in .a.b.c, then it must be the path name for a window; otherwise it may be an arbitrary string. Each window has an associated list of tags, and a binding applies to a particular window if its tag is among those specified for the
window. Although the \textit{bindtags} command may be used to assign an arbitrary set of binding tags to a window, the default binding tags provide the following behavior:

" If a tag is the name of an internal window the binding applies to that window.
" If the tag is the name of a toplevel window the binding applies to the toplevel window and all its
internal windows.
" If the tag is the name of a class of widgets, such as \textbf{Button}, the binding applies to all widgets in that
class;
" If \textit{tag} has the value \textbf{all}, the binding applies to all windows in the application.

\textbf{EVENT PATTERNS}

The \textit{sequence} argument specifies a sequence of one or more event patterns, with optional white space between
the patterns. Each event pattern may take one of three forms. In the simplest case it is a single printing ASCII
character, such as a or [. The character may not be a space character or the character <. This form of pattern
matches a \textbf{KeyPress} event for the particular character. The second form of pattern is longer but more general.
It has the following syntax:

\begin{verbatim}
<modifier-modifier-type-detail>
\end{verbatim}

The entire event pattern is surrounded by angle brackets. Inside the angle brackets are zero or more modifiers,
an event type, and an extra piece of information (\textit{detail}) identifying a particular button or keysym. Any of the
fields may be omitted, as long as at least one of \textit{type} and \textit{detail} is present. The fields must be separated by
white space or dashes.

The third form of pattern is used to specify a user–defined, named virtual event. It has the following syntax:

\begin{verbatim}
<<name>>
\end{verbatim}

The entire virtual event pattern is surrounded by double angle brackets. Inside the angle brackets is the
user–defined name of the virtual event. Modifiers, such as \textbf{Shift} or \textbf{Control}, may not be combined with a
virtual event to modify it. Bindings on a virtual event may be created before the virtual event is defined, and if
the definition of a virtual event changes dynamically, all windows bound to that virtual event will respond
immediately to the new definition.

Some widgets (e.g. \textbf{menu} and \textbf{text}) issue virtual events when their internal state is updated in some ways.
Please see the manual page for each widget for details.

\textbf{MODIFIERS}

Modifiers consist of any of the following values:

\begin{verbatim}
ControlMod2, M2
ShiftMod3, M3
LockMod4, M4
Button1, B1Mod5, M5
Button2, B2Meta, M
Button3, B3Alt
\end{verbatim}
Where more than one value is listed, separated by commas, the values are equivalent. Most of the modifiers have the obvious X meanings. For example, Button1 requires that button 1 be depressed when the event occurs. For a binding to match a given event, the modifiers in the event must include all of those specified in the event pattern. An event may also contain additional modifiers not specified in the binding. For example, if button 1 is pressed while the shift and control keys are down, the pattern <Control–Button–1> will match the event, but <Mod1–Button–1> will not. If no modifiers are specified, then any combination of modifiers may be present in the event.

Meta and M refer to whichever of the M1 through M5 modifiers is associated with the Meta key(s) on the keyboard (keysyms Meta_R and Meta_L). If there are no Meta keys, or if they are not associated with any modifiers, then Meta and M will not match any events. Similarly, the Alt modifier refers to whichever modifier is associated with the alt key(s) on the keyboard (keysyms Alt_L and Alt_R).

The Double, Triple and Quadruple modifiers are a convenience for specifying double mouse clicks and other repeated events. They cause a particular event pattern to be repeated 2, 3 or 4 times, and also place a time and space requirement on the sequence: for a sequence of events to match a Double, Triple or Quadruple pattern, all of the events must occur close together in time and without substantial mouse motion in between. For example, <Double–Button–1> is equivalent to <Button–1><Button–1> with the extra time and space requirement.

**EVENT TYPES**

The type field may be any of the standard X event types, with a few extra abbreviations. The type field will also accept a couple non–standard X event types that were added to better support the Macintosh and Windows platforms. Below is a list of all the valid types; where two names appear together, they are synonyms.

- Activate
- Destroy
- Map
- ButtonPress, Button
- Enter
- MapRequest
- ButtonRelease
- Expose
- Motion
- Circulate
- FocusIn
- MouseWheel
- CirculateRequest
- FocusOut
- Property
- Colormap
- Gravity
- Reparent
- Configure
- KeyPress, Key
- ResizeRequest
- ConfigureRequest
- KeyRelease
- Unmap
- Create
- Leave
- Visibility
- Deactivate

Most of the above events have the same fields and behaviors as events in the X Windowing system. You can find more detailed descriptions of these events in any X window programming book. A couple of the events are extensions to the X event system to support features unique to the Macintosh and Windows platforms. We provide a little more detail on these events here. These include:
Activate, Deactivate

These two events are sent to every sub-window of a toplevel when they change state. In addition to the focus Window, the Macintosh platform and Windows platforms have a notion of an active window (which often has but is not required to have the focus). On the Macintosh, widgets in the active window have a different appearance than widgets in deactive windows. The Activate event is sent to all the sub-windows in a toplevel when it changes from being deactive to active. Likewise, the Deactive event is sent when the window's state changes from active to deactive. There are no useful percent substitutions you would make when binding to these events.

MouseWheel

Some mice on the Windows platform support a mouse wheel which is used for scrolling documents without using the scrollbars. By rolling the wheel, the system will generate MouseWheel events that the application can use to scroll. On Windows, the event is always routed to the window that currently has focus (like Key events.) On Mac OS X, the event is routed to the window under the pointer. When the event is received you can use the %D substitution to get the delta field for the event, which is an integer value describing how the mouse wheel has moved. The smallest value for which the system will report is defined by the OS. On Windows 95 & 98 machines this value is at least 120 before it is reported. However, higher resolution devices may be available in the future. On Mac OS X, the value is not scaled by 120, but a value of 1 corresponds to roughly one text line. The sign of the value determines which direction your widget should scroll. Positive values should scroll up and negative values should scroll down.

KeyPress, KeyRelease

The KeyPress and KeyRelease events are generated whenever a key is pressed or released. KeyPress and KeyRelease events are sent to the window which currently has the keyboard focus.

ButtonPress, ButtonRelease, Motion

The ButtonPress and ButtonRelease events are generated when the user presses or releases a mouse button. Motion events are generated whenever the pointer is moved. ButtonPress, ButtonRelease, and Motion events are normally sent to the window containing the pointer.

When a mouse button is pressed, the window containing the pointer automatically obtains a temporary pointer grab. Subsequent ButtonPress, ButtonRelease, and Motion events will be sent to that window, regardless of which window contains the pointer, until all buttons have been released.

Configure

A Configure event is sent to a window whenever its size, position, or border width changes, and sometimes when it has changed position in the stacking order.

Map, Unmap

The Map and Unmap events are generated whenever the mapping state of a window changes.

Windows are created in the unmapped state. Top-level windows become mapped when they transition to the normal state, and are unmapped in the withdrawn and iconic states. Other windows become mapped when they are placed under control of a geometry manager (for example pack or grid).

A window is viewable only if it and all of its ancestors are mapped. Note that geometry managers typically do not map their children until they have been mapped themselves, and unmap all children when they become unmapped; hence in Tk Map and Unmap events indicate whether or not a window
Visibility
A window is said to be *obscured* when another window above it in the stacking order fully or partially overlaps it. *Visibility* events are generated whenever a window's obscurity state changes; the *state* field (%s) specifies the new state.

Expose
An *Expose* event is generated whenever all or part of a window should be redrawn (for example, when a window is first mapped or if it becomes unobscured). It is normally not necessary for client applications to handle *Expose* events, since Tk handles them internally.

Destroy
A *Destroy* event is delivered to a window when it is destroyed.

When the *Destroy* event is delivered to a widget, it is in a "half−dead" state: the widget still exists, but most operations on it will fail.

FocusIn, FocusOut
The *FocusIn* and *FocusOut* events are generated whenever the keyboard focus changes. A *FocusOut* event is sent to the old focus window, and a *FocusIn* event is sent to the new one.

In addition, if the old and new focus windows do not share a common parent, "virtual crossing" focus events are sent to the intermediate windows in the hierarchy. Thus a *FocusIn* event indicates that the target window or one of its descendants has acquired the focus, and a *FocusOut* event indicates that the focus has been changed to a window outside the target window's hierarchy.

The keyboard focus may be changed explicitly by a call to *focus*, or implicitly by the window manager.

Enter, Leave
An *Enter* event is sent to a window when the pointer enters that window, and a *Leave* event is sent when the pointer leaves it.

If there is a pointer grab in effect, *Enter* and *Leave* events are only delivered to the window owning the grab.

In addition, when the pointer moves between two windows, *Enter* and *Leave* "virtual crossing" events are sent to intermediate windows in the hierarchy in the same manner as for *FocusIn* and *FocusOut* events.

Property
A *Property* event is sent to a window whenever an X property belonging to that window is changed or deleted. *Property* events are not normally delivered to Tk applications as they are handled by the Tk core.

Colormap
A *Colormap* event is generated whenever the colormap associated with a window has been changed, installed, or uninstalled.

Widgets may be assigned a private colormap by specifying a −*colormap* option; the window manager is responsible for installing and uninstalling colormaps as necessary.
Note that Tk provides no useful details for this event type.

MapRequest, CirculateRequest, ResizeRequest, ConfigureRequest, Create

These events are not normally delivered to Tk applications. They are included for completeness, to make it possible to write X11 window managers in Tk. (These events are only delivered when a client has selected SubstructureRedirectMask on a window; the Tk core does not use this mask.)

Gravity, Reparent, Circulate

The events Gravity and Reparent are not normally delivered to Tk applications. They are included for completeness.

A Circulate event indicates that the window has moved to the top or to the bottom of the stacking order as a result of an XCirculateSubwindows protocol request. Note that the stacking order may be changed for other reasons which do not generate a Circulate event, and that Tk does not use XCirculateSubwindows() internally. This event type is included only for completeness; there is no reliable way to track changes to a window's position in the stacking order.

EVENT DETAILS

The last part of a long event specification is detail. In the case of a ButtonPress or ButtonRelease event, it is the number of a button (1–5). If a button number is given, then only an event on that particular button will match; if no button number is given, then an event on any button will match. Note: giving a specific button number is different than specifying a button modifier; in the first case, it refers to a button being pressed or released, while in the second it refers to some other button that is already depressed when the matching event occurs. If a button number is given then type may be omitted: if will default to ButtonPress. For example, the specifier <1> is equivalent to <ButtonPress−1>.

If the event type is KeyPress or KeyRelease, then detail may be specified in the form of an X keysym. Keysyms are textual specifications for particular keys on the keyboard; they include all the alphanumeric ASCII characters (e.g. `a" is the keysym for the ASCII character `a"), plus descriptions for non–alphanumeric characters (` comma" is the keysym for the comma character), plus descriptions for all the non–ASCII keys on the keyboard (` Shift_L" is the keysym for the left shift key, and ` F1" is the keysym for the F1 function key, if it exists). The complete list of keysyms is not presented here; it is available in other X documentation and may vary from system to system. If necessary, you can use the %K notation described below to print out the keysym name for a particular key. If a keysym detail is given, then the type field may be omitted; it will default to KeyPress. For example, <Control–comma> is equivalent to <Control–KeyPress–comma>.

BINDING SCRIPTS AND SUBSTITUTIONS

The script argument to bind is a Tcl script, which will be executed whenever the given event sequence occurs. Command will be executed in the same interpreter that the bind command was executed in, and it will run at global level (only global variables will be accessible). If script contains any % characters, then the script will not be executed directly. Instead, a new script will be generated by replacing each %, and the character following it, with information from the current event. The replacement depends on the character following the %, as defined in the list below. Unless otherwise indicated, the replacement string is the decimal value of the given field from the current event. Some of the substitutions are only valid for certain types of events; if they
are used for other types of events the value substituted is undefined.

%%
Replaced with a single percent.

%#
The number of the last client request processed by the server (the serial field from the event). Valid for all event types.

%a
The above field from the event, formatted as a hexadecimal number. Valid only for Configure events. Indicates the sibling window immediately below the receiving window in the stacking order, or 0 if the receiving window is at the bottom.

%b
The number of the button that was pressed or released. Valid only for ButtonPress and ButtonRelease events.

%c
The count field from the event. Valid only for Expose events. Indicates that there are count pending Expose events which have not yet been delivered to the window.

%d
The detail field from the event. The %d is replaced by a string identifying the detail. For Enter, Leave, FocusIn, and FocusOut events, the string will be one of the following:

- NotifyAncestor
- NotifyNonlinearVirtual
- NotifyDetailNone
- NotifyPointer
- NotifyInferior
- NotifyPointerRoot
- NotifyNonlinear
- NotifyVirtual

For ConfigureRequest events, the string will be one of:

- Above
- Opposite
- Below
- None
- BottomIf
- TopIf

For events other than these, the substituted string is undefined.

%f
The focus field from the event (0 or 1). Valid only for Enter and Leave events. 1 if the receiving window is the focus window or a descendant of the focus window, 0 otherwise.

%h
The height field from the event. Valid for the Configure, ConfigureRequest, Create, ResizeRequest, and Expose events. Indicates the new or requested height of the window.

%i
The window field from the event, represented as a hexadecimal integer. Valid for all event types.

%k
The keycode field from the event. Valid only for KeyPress and KeyRelease events.

%m
The mode field from the event. The substituted string is one of NotifyNormal, NotifyGrab, NotifyUngrab, or NotifyWhileGrabbed. Valid only for Enter, FocusIn, FocusOut, and Leave events.
The override_redirect field from the event. Valid only for Map, Reparent, and Configure events.

The place field from the event, substituted as one of the strings PlaceOnTop or PlaceOnBottom. Valid only for Circulate and CirculateRequest events.

The state field from the event. For ButtonPress, ButtonRelease, Enter, KeyPress, KeyRelease, Leave, and Motion events, a decimal string is substituted. For Visibility, one of the strings VisibilityUnobscured, VisibilityPartiallyObscured, and VisibilityFullyObscured is substituted. For Property events, substituted with either the string NewValue (indicating that the property has been created or modified) or Delete (indicating that the property has been removed).

The time field from the event. This is the X server timestamp (typically the time since the last server reset) in milliseconds, when the event occurred. Valid for most events.

The width field from the event. Indicates the new or requested width of the window. Valid only for Configure, ConfigureRequest, Create, ResizeRequest, and Expose events.

The x and y fields from the event. For ButtonPress, ButtonRelease, Motion, KeyPress, KeyRelease, and MouseWheel events, %x and %y indicate the position of the mouse pointer relative to the receiving window. For Enter and Leave events, the position where the mouse pointer crossed the window, relative to the receiving window. For Configure and Create requests, the x and y coordinates of the window relative to its parent window.

Substitutes the UNICODE character corresponding to the event, or the empty string if the event doesn't correspond to a UNICODE character (e.g. the shift key was pressed). XmbLookupString (or XLookupString when input method support is turned off) does all the work of translating from the event to a UNICODE character. Valid only for KeyPress and KeyRelease events.

The border_width field from the event. Valid only for Configure, ConfigureRequest, and Create events.

This reports the delta value of a MouseWheel event. The delta value represents the rotation units the mouse wheel has been moved. On Windows 95 & 98 systems the smallest value for the delta is 120. Future systems may support higher resolution values for the delta. The sign of the value represents the direction the mouse wheel was scrolled.

The send_event field from the event. Valid for all event types. 0 indicates that this is a `normal` event, 1 indicates that it is a `synthetic` event generated by SendEvent.

The keysym corresponding to the event, substituted as a textual string. Valid only for KeyPress and KeyRelease events.

The keysym corresponding to the event, substituted as a decimal number. Valid only for KeyPress and KeyRelease events.
The name of the property being updated or deleted (which may be converted to an XAtom using \texttt{winfo atom}.) Valid only for \texttt{Property} events.

The \texttt{root} window identifier from the event. Valid only for events containing a \texttt{root} field.

The \texttt{subwindow} window identifier from the event, formatted as a hexadecimal number. Valid only for events containing a \texttt{subwindow} field.

The \texttt{type} field from the event. Valid for all event types.

The path name of the window to which the event was reported (the \texttt{window} field from the event). Valid for all event types.

The \texttt{x\_root} field from the event. If a virtual–root window manager is being used then the substituted value is the corresponding \texttt{x}–coordinate in the virtual root. Valid only for \texttt{ButtonPress}, \texttt{ButtonRelease}, \texttt{KeyPress}, \texttt{KeyRelease}, and \texttt{Motion} events. Same meaning as \texttt{%x}, except relative to the (virtual) root window.

The \texttt{y\_root} field from the event. If a virtual–root window manager is being used then the substituted value is the corresponding \texttt{y}–coordinate in the virtual root. Valid only for \texttt{ButtonPress}, \texttt{ButtonRelease}, \texttt{KeyPress}, \texttt{KeyRelease}, and \texttt{Motion} events. Same meaning as \texttt{%y}, except relative to the (virtual) root window.

The replacement string for a \texttt{%}–replacement is formatted as a proper Tcl list element. This means that it will be surrounded with braces if it contains spaces, or special characters such as $ and \{ may be preceded by backslashes. This guarantees that the string will be passed through the Tcl parser when the binding script is evaluated. Most replacements are numbers or well–defined strings such as \texttt{Above}; for these replacements no special formatting is ever necessary. The most common case where reformatting occurs is for the \texttt{%A} substitution. For example, if \texttt{script} is

\begin{verbatim}
insert %A
\end{verbatim}

and the character typed is an open square bracket, then the script actually executed will be

\begin{verbatim}
insert \[
\end{verbatim}

This will cause the \texttt{insert} to receive the original replacement string (open square bracket) as its first argument. If the extra backslash hadn't been added, Tcl would not have been able to parse the script correctly.

**MULTIPLE MATCHES**

It is possible for several bindings to match a given X event. If the bindings are associated with different \texttt{tag}’s, then each of the bindings will be executed, in order. By default, a binding for the widget will be executed first, followed by a class binding, a binding for its toplevel, and an \texttt{all} binding. The \texttt{bindtags} command may be used to change this order for a particular window or to associate additional binding tags with the window.
The `continue` and `break` commands may be used inside a binding script to control the processing of matching scripts. If `continue` is invoked, then the current binding script is terminated but Tk will continue processing binding scripts associated with other tag's. If the `break` command is invoked within a binding script, then that script terminates and no other scripts will be invoked for the event.

If more than one binding matches a particular event and they have the same `tag`, then the most specific binding is chosen and its script is evaluated. The following tests are applied, in order, to determine which of several matching sequences is more specific: (a) an event pattern that specifies a specific button or key is more specific than one that doesn't; (b) a longer sequence (in terms of number of events matched) is more specific than a shorter sequence; (c) if the modifiers specified in one pattern are a subset of the modifiers in another pattern, then the pattern with more modifiers is more specific. (d) a virtual event whose physical pattern matches the sequence is less specific than the same physical pattern that is not associated with a virtual event. (e) given a sequence that matches two or more virtual events, one of the virtual events will be chosen, but the order is undefined.

If the matching sequences contain more than one event, then tests (c)–(e) are applied in order from the most recent event to the least recent event in the sequences. If these tests fail to determine a winner, then the most recently registered sequence is the winner.

If there are two (or more) virtual events that are both triggered by the same sequence, and both of those virtual events are bound to the same window tag, then only one of the virtual events will be triggered, and it will be picked at random:

```tcl
event add <<Paste>> <Control-y>
event add <<Paste>> <Button-2>
event add <<Scroll>> <Button-2>
bind Entry <<Paste>> {puts Paste}
bind Entry <<Scroll>> {puts Scroll}
```

If the user types Control–y, the `<<Paste>>` binding will be invoked, but if the user presses button 2 then one of either the `<<Paste>>` or the `<<Scroll>>` bindings will be invoked, but exactly which one gets invoked is undefined.

If an X event does not match any of the existing bindings, then the event is ignored. An unbound event is not considered to be an error.

**MULTI-EVENT SEQUENCES AND IGNORED EVENTS**

When a `sequence` specified in a `bind` command contains more than one event pattern, then its script is executed whenever the recent events (leading up to and including the current event) match the given sequence. This means, for example, that if button 1 is clicked repeatedly the sequence `<<Double-ButtonPress-1>>` will match each button press but the first. If extraneous events that would prevent a match occur in the middle of an event sequence then the extraneous events are ignored unless they are `KeyPress` or `ButtonPress` events. For example, `<<Double-ButtonPress-1>>` will match a sequence of presses of button 1, even though there will be `ButtonRelease` events (and possibly `Motion` events) between the `ButtonPress` events. Furthermore, a `KeyPress` event may be preceded by any number of other `KeyPress`
events for modifier keys without the modifier keys preventing a match. For example, the event sequence \texttt{aB} will match a press of the \texttt{a} key, a release of the \texttt{a} key, a press of the \texttt{Shift} key, and a press of the \texttt{b} key: the press of \texttt{Shift} is ignored because it is a modifier key. Finally, if several \texttt{Motion} events occur in a row, only the last one is used for purposes of matching binding sequences.

**ERRORS**

If an error occurs in executing the script for a binding then the \texttt{bgerror} mechanism is used to report the error. The \texttt{bgerror} command will be executed at global level (outside the context of any Tcl procedure).

**EXAMPLES**

Arrange for a string describing the motion of the mouse to be printed out when the mouse is double-clicked:

```tcl
bind . <Double-1> {
    puts "hi from (%x,%y)"
}
```

A little GUI that displays what the keysym name of the last key pressed is:

```tcl
set keysym "Press any key"
pack [label .l -textvariable keysym -padx 2m -pady 1m]
bind . <Key> {
    set keysym "You pressed %K"
}
```

**SEE ALSO**

\texttt{bgerror}, \texttt{bindtags}, \texttt{event}, \texttt{focus}, \texttt{grab}, \texttt{keysyms}

**KEYWORDS**

\texttt{binding, event}
bindtags

NAME

bindtags – Determine which bindings apply to a window, and order of evaluation

SYNOPSIS

bindtags window ?tagList?

DESCRIPTION

When a binding is created with the bind command, it is associated either with a particular window such as .a.b.c, a class name such as Button, the keyword all, or any other string. All of these forms are called binding tags. Each window contains a list of binding tags that determine how events are processed for the window. When an event occurs in a window, it is applied to each of the window's tags in order: for each tag, the most specific binding that matches the given tag and event is executed. See the bind command for more information on the matching process.

By default, each window has four binding tags consisting of the name of the window, the window's class name, the name of the window's nearest toplevel ancestor, and all, in that order. Toplevel windows have only three tags by default, since the toplevel name is the same as that of the window. The bindtags command allows the binding tags for a window to be read and modified.

If bindtags is invoked with only one argument, then the current set of binding tags for window is returned as a list. If the tagList argument is specified to bindtags, then it must be a proper list; the tags for window are changed to the elements of the list. The elements of tagList may be arbitrary strings; however, any tag starting with a dot is treated as the name of a window; if no window by that name exists at the time an event is processed, then the tag is ignored for that event. The order of the elements in tagList determines the order in which binding scripts are executed in response to events. For example, the command

bindtags .b {all . Button .b}

reverses the order in which binding scripts will be evaluated for a button named .b so that all bindings are invoked first, following by bindings for .b's toplevel ("."), followed by class bindings, followed by bindings for .b. If tagList is an empty list then the binding tags for window are returned to the default state described above.

The bindtags command may be used to introduce arbitrary additional binding tags for a window, or to remove standard tags. For example, the command

bindtags .b { .b TrickyButton . all}

replaces the Button tag for .b with TrickyButton. This means that the default widget bindings for buttons, which are associated with the Button tag, will no longer apply to .b, but any bindings associated with
TrickyButton (perhaps some new button behavior) will apply.

EXAMPLE

If you have a set of nested frame widgets and you want events sent to a button widget to also be delivered to all the widgets up to the current toplevel (in contrast to Tk's default behavior, where events are not delivered to those intermediate windows) to make it easier to have accelerators that are only active for part of a window, you could use a helper procedure like this to help set things up:

```tcl
proc setupBindtagsForTreeDelivery {widget} {
    set tags [list $widget [winfo class $widget]]
    set w $widget
    set t [winfo toplevel $w]
    while {$w ne $t} {
        set w [winfo parent $w]
        lappend tags $w
    }
    lappend tags all
    bindtags $widget $tags
}
```

SEE ALSO

bind

KEYWORDS

binding, event, tag

Copyright ' 1990 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

bitmap – Images that display two colors

SYNOPSIS

image create bitmap ?name? ?options?

DESCRIPTION

CREATING BITMAPS

−background color
−data string
−file name
−foreground color
−maskdata string
−maskfile name

IMAGE COMMAND

imageName cget option
imageName configure ?option? ?value option value ...?

KEYWORDS

NAME

bitmap – Images that display two colors

SYNOPSIS

image create bitmap ?name? ?options?

DESCRIPTION

A bitmap is an image whose pixels can display either of two colors or be transparent. A bitmap image is defined by four things: a background color, a foreground color, and two bitmaps, called the source and the mask. Each of the bitmaps specifies 0/1 values for a rectangular array of pixels, and the two bitmaps must have the same dimensions. For pixels where the mask is zero, the image displays nothing, producing a transparent effect. For other pixels, the image displays the foreground color if the source data is one and the background color if the source data is zero.

CREATING BITMAPS

Like all images, bitmaps are created using the image create command. Bitmaps support the following options:

−background color
Specifies a background color for the image in any of the standard ways accepted by Tk. If this option is set to an empty string then the background pixels will be transparent. This effect is achieved by using the source bitmap as the mask bitmap, ignoring any \texttt{\--maskdata} or \texttt{\--maskfile} options.

\texttt{\--data \texttt{string}}

Specifies the contents of the source bitmap as a string. The string must adhere to X11 bitmap format (e.g., as generated by the \texttt{bitmap} program). If both the \texttt{\--data} and \texttt{\--file} options are specified, the \texttt{\--data} option takes precedence.

\texttt{\--file \texttt{name}}

\texttt{name} gives the name of a file whose contents define the source bitmap. The file must adhere to X11 bitmap format (e.g., as generated by the \texttt{bitmap} program).

\texttt{\--foreground \texttt{color}}

Specifies a foreground color for the image in any of the standard ways accepted by Tk.

\texttt{\--maskdata \texttt{string}}

Specifies the contents of the mask as a string. The string must adhere to X11 bitmap format (e.g., as generated by the \texttt{bitmap} program). If both the \texttt{\--maskdata} and \texttt{\--maskfile} options are specified, the \texttt{\--maskdata} option takes precedence.

\texttt{\--maskfile \texttt{name}}

\texttt{name} gives the name of a file whose contents define the mask. The file must adhere to X11 bitmap format (e.g., as generated by the \texttt{bitmap} program).

\section*{IMAGE COMMAND}

When a bitmap image is created, Tk also creates a new command whose name is the same as the image. This command may be used to invoke various operations on the image. It has the following general form:

\begin{verbatim}
imageName option ?arg arg ...? 
\end{verbatim}

\textit{Option} and the \textit{args} determine the exact behavior of the command. The following commands are possible for bitmap images:

\begin{verbatim}
imageName eget option
\end{verbatim}

Returns the current value of the configuration option given by \textit{option}. \textit{Option} may have any of the values accepted by the \texttt{image create bitmap} command.

\begin{verbatim}
imageName configure ?option? ?value option value ...? 
\end{verbatim}

Query or modify the configuration options for the image. If no \textit{option} is specified, returns a list describing all of the available options for \textit{imageName} (see \texttt{TkConfigureInfo} for information on the format of this list). If \textit{option} is specified with no \textit{value}, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no \textit{option} is specified). If one or more \textit{option}=\textit{value} pairs are specified, then the command modifies the given option(s) to have the given value(s); in this case the command returns an empty string. \textit{Option} may have any of the values accepted by the \texttt{image create bitmap} command.
KEYWORDS

bitmap, image
NAME

button – Create and manipulate button widgets

SYNOPSIS

button pathName ?options?

STANDARD OPTIONS

- activebackground, activeBackground, Foreground
- activeforeground, activeForeground, Background
- anchor, anchor, Anchor
- background or -bg, background, Background
- bitmap, bitmap, Bitmap
- borderwidth or -bd, borderWidth, BorderWidth
- compound, compound, Compound
- cursor, cursor, Cursor
- disabledforeground, disabledForeground, DisabledForeground
- font, font, Font
- foreground or -fg, foreground, Foreground
- highlightbackground, highlightBackground, HighlightBackground
- highlightcolor, highlightColor, HighlightColor
- highlightthickness, highlightThickness, HighlightThickness
- image, image, Image
- justify, justify, Justify
- padx, padX, Pad
- pady, padY, Pad
- relief, relief, Relief
- repeatdelay, repeatDelay, RepeatDelay
- repeatinterval, repeatInterval, RepeatInterval
- takefocus, takeFocus, TakeFocus
- text, text, Text
- textvariable, textVariable, Variable
- underline, underline, Underline
- wraplength, wrapLength, WrapLength

WIDGET-SPECIFIC OPTIONS

- command, command, Command
- default, default, Default
- height, height, Height
- overrelief, overRelief, OverRelief
- state, state, State
- width, width, Width

DESCRIPTION

WIDGET COMMAND

pathName cget option
**NAME**

button – Create and manipulate button widgets

**SYNOPSIS**

`button pathName ?options?`

**STANDARD OPTIONS**

- `activebackground`, `activeBackground`, `Foreground`
- `activeforeground`, `activeForeground`, `Background`
- `anchor`, `anchor`, `Anchor`
- `background or –bg`, `background`, `Background`
- `bitmap`, `bitmap`, `Bitmap`
- `borderwidth or –bd`, `borderWidth`, `BorderWidth`
- `compound`, `compound`, `Compound`
- `cursor`, `cursor`, `Cursor`
- `disabledforeground`, `disabledForeground`, `DisabledForeground`
- `font`, `font`, `Font`
- `foreground or –fg`, `foreground`, `Foreground`
- `highlightbackground`, `highlightBackground`, `HighlightBackground`
- `highlightcolor`, `highlightColor`, `HighlightColor`
- `highlightthickness`, `highlightThickness`, `HighlightThickness`
- `image`, `image`, `Image`
- `justify`, `justify`, `Justify`
- `padx`, `padX`, `Pad`
- `pady`, `padY`, `Pad`
- `relief`, `relief`, `Relief`
- `repeatdelay`, `repeatDelay`, `RepeatDelay`
- `repeatinterval`, `repeatInterval`, `RepeatInterval`
- `takefocus`, `takeFocus`, `TakeFocus`
- `text`, `text`, `Text`
- `textvariable`, `textVariable`, `Variable`
- `underline`, `underline`, `Underline`
- `wraplength`, `wrapLength`, `WrapLength`
WIDGET–SPECIFIC OPTIONS

Command–Line Name: –command
Database Name: command
Database Class: Command

Specifies a Tcl command to associate with the button. This command is typically invoked when mouse button 1 is released over the button window.

Command–Line Name: –default
Database Name: default
Database Class: Default

Specifies one of three states for the default ring: normal, active, or disabled. In active state, the button is drawn with the platform specific appearance for a default button. In normal state, the button is drawn with the platform specific appearance for a non–default button, leaving enough space to draw the default button appearance. The normal and active states will result in buttons of the same size. In disabled state, the button is drawn with the non–default button appearance without leaving space for the default appearance. The disabled state may result in a smaller button than the active state.

Command–Line Name: –height
Database Name: height
Database Class: Height

Specifies a desired height for the button. If an image or bitmap is being displayed in the button then the value is in screen units (i.e. any of the forms acceptable to Tk_GetPixels); for text it is in lines of text. If this option isn't specified, the button's desired height is computed from the size of the image or bitmap or text being displayed in it.

Command–Line Name: –overrelief
Database Name: overRelief
Database Class: OverRelief

Specifies an alternative relief for the button, to be used when the mouse cursor is over the widget. This option can be used to make toolbar buttons, by configuring –relief flat –overrelief raised. If the value of this option is the empty string, then no alternative relief is used when the mouse cursor is over the button. The empty string is the default value.

Command–Line Name: –state
Database Name: state
Database Class: State

Specifies one of three states for the button: normal, active, or disabled. In normal state the button is displayed using the foreground and background options. The active state is typically used when the pointer is over the button. In active state the button is displayed using the activeForeground and activeBackground options. Disabled state means that the button should be insensitive: the default bindings will refuse to activate the widget and will ignore mouse button presses. In this state the disabledForeground and background options determine how the button is displayed.

Command–Line Name: –width
Database Name: width
Database Class: Width

Specifies a desired width for the button. If an image or bitmap is being displayed in the button then the value is in screen units (i.e. any of the forms acceptable to Tk_GetPixels); for text it is in.
characters. If this option isn't specified, the button's desired width is computed from the size of the image or bitmap or text being displayed in it.

DESCRIPTION

The `button` command creates a new window (given by the `pathName` argument) and makes it into a button widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the button such as its colors, font, text, and initial relief. The `button` command returns its `pathName` argument. At the time this command is invoked, there must not exist a window named `pathName`, but `pathName`'s parent must exist.

A button is a widget that displays a textual string, bitmap or image. If text is displayed, it must all be in a single font, but it can occupy multiple lines on the screen (if it contains newlines or if wrapping occurs because of the `wrapLength` option) and one of the characters may optionally be underlined using the `underline` option. It can display itself in either of three different ways, according to the `state` option; it can be made to appear raised, sunken, or flat; and it can be made to flash. When a user invokes the button (by pressing mouse button 1 with the cursor over the button), then the Tcl command specified in the `−command` option is invoked.

WIDGET COMMAND

The `button` command creates a new Tcl command whose name is `pathName`. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

`Option` and the `args` determine the exact behavior of the command. The following commands are possible for button widgets:

```
pathName cget option
```

Returns the current value of the configuration option given by `option`. `Option` may have any of the values accepted by the `button` command.

```
pathName configure ?option? ?value option value ...?
```

Query or modify the configuration options of the widget. If no `option` is specified, returns a list describing all of the available options for `pathName` (see `Tk_ConfigureInfo` for information on the format of this list). If `option` is specified with no `value`, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no `option` is specified). If one or more `option−value` pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. `Option` may have any of the values accepted by the `button` command.

```
pathName flash
```

Flash the button. This is accomplished by redisplaying the button several times, alternating between active and normal colors. At the end of the flash the button is left in the same normal/active state as when the command was invoked. This command is ignored if the button's state is `disabled`.

```
pathName invoke
```

DESCRIPTION 382
Invoke the Tcl command associated with the button, if there is one. The return value is the return value from the Tcl command, or an empty string if there is no command associated with the button. This command is ignored if the button’s state is disabled.

DEFAULT BINDINGS

Tk automatically creates class bindings for buttons that give them default behavior:

[1] A button activates whenever the mouse passes over it and deactivates whenever the mouse leaves the button. Under Windows, this binding is only active when mouse button 1 has been pressed over the button.

[2] A button's relief is changed to sunken whenever mouse button 1 is pressed over the button, and the relief is restored to its original value when button 1 is later released.

[3] If mouse button 1 is pressed over a button and later released over the button, the button is invoked. However, if the mouse is not over the button when button 1 is released, then no invocation occurs.

[4] When a button has the input focus, the space key causes the button to be invoked.

If the button's state is disabled then none of the above actions occur: the button is completely non-responsive.

The behavior of buttons can be changed by defining new bindings for individual widgets or by redefining the class bindings.

KEYWORDS

button, widget

Copyright ' 1990-1994 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

canvas – Create and manipulate canvas widgets

SYNOPSIS

canvas pathName ?options?

STANDARD OPTIONS

- background or -bg, background, Background
- borderwidth or -bd, borderWidth, BorderWidth
- cursor, cursor, Cursor
- highlightbackground, highlightBackground, HighlightBackground
- highlightcolor, highlightColor, HighlightColor
- highlightthickness, highlightThickness, HighlightThickness
- insertbackground, insertBackground, Foreground
- insertborderwidth, insertBorderWidth, BorderWidth
- insertofftime, insertOffTime, OffTime
- insertontime, insertOnTime, OnTime
- insertwidth, insertWidth, InsertWidth
- relief, relief, Relief
- selectbackground, selectBackground, Foreground
- selectborderwidth, selectBorderWidth, BorderWidth
- selectforeground, selectForeground, Background
- state
- takefocus, takeFocus, TakeFocus
- xscrollcommand, xScrollCommand, ScrollCommand
- yscrollcommand, yScrollCommand, ScrollCommand

WIDGET–SPECIFIC OPTIONS

- closeenough, closeEnough, CloseEnough
- confine, confine, Confine
- height, height, Height
- scrollregion, scrollRegion, ScrollRegion
- state, state, State
- width, width, width
- xscrollincrement, xScrollIncrement, ScrollIncrement
- yscrollincrement, yScrollIncrement, ScrollIncrement

INTRODUCTION

DISPLAY LIST
ITEM IDS AND TAGS
COORDINATES
TRANSFORMATIONS
INDICES

number

end
insert
sel.first
sel.last
@x,y

DASH PATTERNS
WIDGET COMMAND

pathName addtag tag searchSpec ?arg arg ...?
above tagOrId
all
below tagOrId
closest x y ?halo? ?start?
enclosed x1 y1 x2 y2
overlapping x1 y1 x2 y2
withtag tagOrId
pathName bbox ?tagOrId tagOrId tagOrId ...
pathName bind tagOrId ?sequence? ?command?
pathName canvax screen x y ?gridspacing?
pathName canvasy screen y ?gridspacing?
pathName cget option
pathName configure ?option? ?value? ?option value ...?
pathName coords tagOrId ?x0 y0 ...
pathName coords tagOrId ?coordList?
pathName create type x y ?x y ... ?option value ...
pathName create type coordList ?option value ...
pathName dchars tagOrId first ?last?
pathName delete ?tagOrId tagOrId ...
pathName dtag tagOrId ?tagToDelete?
pathName find searchCommand ?arg arg ...
pathName focus ?tagOrId?
pathName gettags tagOrId
pathName icursor tagOrId index
pathName index tagOrId index
pathName insert tagOrId beforeThis string
pathName itemcget tagOrId option
pathName itemconfigure tagOrId ?option? ?value? ?option value ...
pathName lower tagOrId ?belowThis?
pathName move tagOrId xAmount yAmount
pathName postscript ?option value option value ...
−colormap varName
−colormode mode
−file fileName
−fontmap varName
−height size
−pageanchor anchor
−pageheight size
-pagewidth size
-pagex position
-pagey position
-rotate boolean
-width size
-x position
-y position

pathName raise tagOrId ?aboveThis?
pathName scale tagOrId xOrigin yOrigin xScale yScale

pathName scan option args
  pathName scan mark x y
  pathName scan dragto x y ?gain?.

pathName select option ?tagOrId arg?
  pathName select adjust tagOrId index
  pathName select clear
  pathName select from tagOrId index
  pathName select item
  pathName select to tagOrId index

pathName type tagOrId

pathName xview ?args?
  pathName xview
  pathName xview moveto fraction
  pathName xview scroll number what

pathName yview ?args?
  pathName yview
  pathName yview moveto fraction
  pathName yview scroll number what

OVERVIEW OF ITEM TYPES
COMMON ITEM OPTIONS
  -dash pattern
  -activedash pattern
  -disableddash pattern
  -dashoffset offset
  -fill color
  -activefill color
  -disabledfill color
  -outline color
  -activeoutline color
  -disabledoutline color
  -offset offset
  -outlinestipple bitmap
  -activeoutlinestipple bitmap
  -disabledoutlinestipple bitmap
  -stipple bitmap
  -activestipple bitmap
-disabledstipple bitmap
-state state
-tags tagList
-width outlineWidth
-activewidth outlineWidth
-disabledwidth outlineWidth

ARC ITEMS
-extent degrees
-start degrees
-style type

BITMAP ITEMS
-anchor anchorPos
-background color
-activebackground bitmap
-disabledbackground bitmap
-bitmap bitmap
-activebitmap bitmap
-disabledbitmap bitmap
-foreground color
-activeforeground bitmap
-disabledforeground bitmap

IMAGE ITEMS
-anchor anchorPos
-image name
-activeimage name
-disabledimage name

LINE ITEMS
-arrow where
-arrowshape shape
-capstyle style
-joinstyle style
-smooth smoothMethod
-splinesteps number

OVAL ITEMS

POLYGON ITEMS
-joinstyle style
-smooth boolean
-splinesteps number

RECTANGLE ITEMS

TEXT ITEMS
-anchor anchorPos
-fontfontName
-justify how
-text string
-width lineLength
WINDOW ITEMS

- **anchor** anchorPos
- **height** pixels
- **width** pixels
- **window** pathName

APPLICATION-DEFINED ITEM TYPES

BINDINGS

CREDITS

SEE ALSO

KEYWORDS

NAME

canvas – Create and manipulate canvas widgets

SYNOPSIS

canvas pathName ?options?

STANDARD OPTIONS

- **background** or **-bg, background, Background**
- **borderwidth** or **-bd, borderWidth, BorderWidth**
- **cursor**
- **highlightbackground** or **highlightBackground, HighlightBackground**
- **highlightcolor** or **highlightColor, HighlightColor**
- **highlightthickness** or **highlightThickness, HighlightThickness**
- **insertbackground** or **insertBackground, Foreground**
- **insertborderwidth** or **insertBorderWidth, BorderWidth**
- **insertofftime** or **insertOffTime, OffTime**
- **insertontime** or **insertOnTime, OnTime**
- **insertwidth** or **insertWidth, InsertWidth**
- **relief** or **relief, Relief**
- **selectbackground** or **selectBackground, Foreground**
- **selectborderwidth** or **selectBorderWidth, BorderWidth**
- **selectforeground** or **selectForeground, Background**
- **state**
- **takefocus** or **takeFocus, TakeFocus**
- **xscrollcommand** or **xScrollCommand, ScrollCommand**
- **yscrollcommand** or **yScrollCommand, ScrollCommand**

WIDGET–SPECIFIC OPTIONS

Command–Line Name: **-closeenough**
Database Name: closeEnough
Database Class: CloseEnough

Specifies a floating-point value indicating how close the mouse cursor must be to an item before it is considered to be "inside" the item. Defaults to 1.0.

Command–Line Name: −confine
Database Name: confine
Database Class: Confine

Specifies a boolean value that indicates whether or not it should be allowable to set the canvas's view outside the region defined by the scrollRegion argument. Defaults to true, which means that the view will be constrained within the scroll region.

Command–Line Name: −height
Database Name: height
Database Class: Height

Specifies a desired window height that the canvas widget should request from its geometry manager. The value may be specified in any of the forms described in the COORDINATES section below.

Command–Line Name: −scrollregion
Database Name: scrollRegion
Database Class: ScrollRegion

Specifies a list with four coordinates describing the left, top, right, and bottom coordinates of a rectangular region. This region is used for scrolling purposes and is considered to be the boundary of the information in the canvas. Each of the coordinates may be specified in any of the forms given in the COORDINATES section below.

Command–Line Name: −state
Database Name: state
Database Class: State

Modifies the default state of the canvas where state may be set to one of: normal, disabled, or hidden. Individual canvas objects all have their own state option which may override the default state. Many options can take separate specifications such that the appearance of the item can be different in different situations. The options that start with active control the appearance when the mouse pointer is over it, while the option starting with disabled controls the appearance when the state is disabled. Canvas items which are disabled will not react to canvas bindings.

Command–Line Name: −width
Database Name: width
Database Class: width

Specifies a desired window width that the canvas widget should request from its geometry manager. The value may be specified in any of the forms described in the COORDINATES section below.

Command–Line Name: −xscrollincrement
Database Name: xScrollIncrement
Database Class: ScrollIncrement

Specifies an increment for horizontal scrolling, in any of the usual forms permitted for screen distances. If the value of this option is greater than zero, the horizontal view in the window will be constrained so that the canvas x coordinate at the left edge of the window is always an even multiple of xScrollIncrement; furthermore, the units for scrolling (e.g., the change in view when the left and right arrows of a scrollbar are selected) will also be xScrollIncrement. If the value of this option is less than or equal to zero, then horizontal scrolling is unconstrained.
Command-Line Name: -yscrollincrement
Database Name: yScrollIncrement
Database Class: ScrollIncrement

Specifies an increment for vertical scrolling, in any of the usual forms permitted for screen distances. If the value of this option is greater than zero, the vertical view in the window will be constrained so that the canvas y coordinate at the top edge of the window is always an even multiple of yScrollIncrement; furthermore, the units for scrolling (e.g., the change in view when the top and bottom arrows of a scrollbar are selected) will also be yScrollIncrement. If the value of this option is less than or equal to zero, then vertical scrolling is unconstrained.

INTRODUCTION

The canvas command creates a new window (given by the pathName argument) and makes it into a canvas widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the canvas such as its colors and 3-D relief. The canvas command returns its pathName argument. At the time this command is invoked, there must not exist a window named pathName, but pathName's parent must exist.

Canvas widgets implement structured graphics. A canvas displays any number of items, which may be things like rectangles, circles, lines, and text. Items may be manipulated (e.g. moved or re-colored) and commands may be associated with items in much the same way that the bind command allows commands to be bound to widgets. For example, a particular command may be associated with the <Button-1> event so that the command is invoked whenever button 1 is pressed with the mouse cursor over an item. This means that items in a canvas can have behaviors defined by the Tcl scripts bound to them.

DISPLAY LIST

The items in a canvas are ordered for purposes of display, with the first item in the display list being displayed first, followed by the next item in the list, and so on. Items later in the display list obscure those that are earlier in the display list and are sometimes referred to as being `on top' of earlier items. When a new item is created it is placed at the end of the display list, on top of everything else. Widget commands may be used to re-arrange the order of the display list.

Window items are an exception to the above rules. The underlying window systems require them always to be drawn on top of other items. In addition, the stacking order of window items is not affected by any of the canvas widget commands; you must use the raise and lower Tk commands instead.

ITEM IDS AND TAGS

Items in a canvas widget may be named in either of two ways: by id or by tag. Each item has a unique identifying number, which is assigned to that item when it is created. The id of an item never changes and id numbers are never re-used within the lifetime of a canvas widget.

Each item may also have any number of tags associated with it. A tag is just a string of characters, and it may take any form except that of an integer. For example, "x123" is OK but "123" isn't. The same tag may be
associated with many different items. This is commonly done to group items in various interesting ways; for
element, all selected items might be given the tag "selected".

The tag all is implicitly associated with every item in the canvas; it may be used to invoke operations on all
the items in the canvas.

The tag current is managed automatically by Tk; it applies to the current item, which is the topmost item
whose drawn area covers the position of the mouse cursor. If the mouse is not in the canvas widget or is not
over an item, then no item has the current tag.

When specifying items in canvas widget commands, if the specifier is an integer then it is assumed to refer to
the single item with that id. If the specifier is not an integer, then it is assumed to refer to all of the items in the
canvas that have a tag matching the specifier. The symbol tagOrId is used below to indicate that an argument
specifies either an id that selects a single item or a tag that selects zero or more items.

tagOrId may contain a logical expressions of tags by using operators: '&&', '||', '^', '!', and parenthesized
subexpressions. For example:

.c find withtag {(a&&!b)||(!a&&b)}

or equivalently:

.c find withtag {a^b}

will find only those items with either "a" or "b" tags, but not both.

Some widget commands only operate on a single item at a time; if tagOrId is specified in a way that names
multiple items, then the normal behavior is for the command to use the first (lowest) of these items in the
display list that is suitable for the command. Exceptions are noted in the widget command descriptions below.

COORDINATES

All coordinates related to canvases are stored as floating-point numbers. Coordinates and distances are
specified in screen units, which are floating-point numbers optionally followed by one of several letters. If no
letter is supplied then the distance is in pixels. If the letter is m then the distance is in millimeters on the
screen; if it is c then the distance is in centimeters; i means inches, and p means printers points (1/72 inch).
Larger y−coordinates refer to points lower on the screen; larger x−coordinates refer to points farther to the
right. Coordinates can be specified either as an even number of parameters, or as a single list parameter
containing an even number of x and y coordinate values.

TRANSFORMATIONS

Normally the origin of the canvas coordinate system is at the upper−left corner of the window containing the
canvas. It is possible to adjust the origin of the canvas coordinate system relative to the origin of the window
using the xview and yview widget commands; this is typically used for scrolling. Canvases do not support
scaling or rotation of the canvas coordinate system relative to the window coordinate system.

Individual items may be moved or scaled using widget commands described below, but they may not be rotated.

Note that the default origin of the canvas's visible area is coincident with the origin for the whole window as that makes bindings using the mouse position easier to work with; you only need to use the `canvasx` and `canvasy` widget commands if you adjust the origin of the visible area. However, this also means that any focus ring (as controlled by the `−highlightthickness` option) and window border (as controlled by the `−borderwidth` option) must be taken into account before you get to the visible area of the canvas.

**INDICES**

Text items support the notion of an index for identifying particular positions within the item. In a similar fashion, line and polygon items support index for identifying, inserting and deleting subsets of their coordinates. Indices are used for commands such as inserting or deleting a range of characters or coordinates, and setting the insertion cursor position. An index may be specified in any of a number of ways, and different types of items may support different forms for specifying indices. Text items support the following forms for an index; if you define new types of text–like items, it would be advisable to support as many of these forms as practical. Note that it is possible to refer to the character just after the last one in the text item; this is necessary for such tasks as inserting new text at the end of the item. Lines and Polygons don't support the insertion cursor and the selection. Their indices are supposed to be even always, because coordinates always appear in pairs.

- **number**
  A decimal number giving the position of the desired character within the text item. 0 refers to the first character, 1 to the next character, and so on. If indexes are odd for lines and polygons, they will be automatically decremented by one. A number less than 0 is treated as if it were zero, and a number greater than the length of the text item is treated as if it were equal to the length of the text item. For polygons, numbers less than 0 or greater then the length of the coordinate list will be adjusted by adding or subtracting the length until the result is between zero and the length, inclusive.

- **end**
  Refers to the character or coordinate just after the last one in the item (same as the number of characters or coordinates in the item).

- **insert**
  Refers to the character just before which the insertion cursor is drawn in this item. Not valid for lines and polygons.

- **sel.first**
  Refers to the first selected character in the item. If the selection isn’t in this item then this form is illegal.

- **sel.last**
  Refers to the last selected character in the item. If the selection isn’t in this item then this form is illegal.

- **@x,y**
  Refers to the character or coordinate at the point given by x and y, where x and y are specified in the...
coordinate system of the canvas. If \( x \) and \( y \) lie outside the coordinates covered by the text item, then they refer to the first or last character in the line that is closest to the given point.

**DASH PATTERNS**

Many items support the notion of a dash pattern for outlines.

The first possible syntax is a list of integers. Each element represents the number of pixels of a line segment. Only the odd segments are drawn using the "outline" color. The other segments are drawn transparent.

The second possible syntax is a character list containing only 5 possible characters \[.,−_\]. The space can be used to enlarge the space between other line elements, and can not occur as the first position in the string. Some examples:

\[-\text{dash} . = -\text{dash} \{2 4\} -\text{dash} _ = -\text{dash} \{6 4\} -\text{dash} . = -\text{dash} \{6 4 2 4\} -\text{dash} _ . = -\text{dash} \{6 4 2 4 2 4\} -\text{dash} \{. \} = -\text{dash} \{2 8\} -\text{dash} , = -\text{dash} \{4 4\}\]

The main difference of this syntax with the previous is that it is shape-conserving. This means that all values in the dash list will be multiplied by the line width before display. This assures that "." will always be displayed as a dot and "-" always as a dash regardless of the line width.

On systems which support only a limited set of dash patterns, the dash pattern will be displayed as the closest dash pattern that is available. For example, on Windows only the first 4 of the above examples are available. The last 2 examples will be displayed identically to the first one.

**WIDGET COMMAND**

The `canvas` command creates a new Tcl command whose name is `pathName`. This command may be used to invoke various operations on the widget. It has the following general form:

\[
\text{pathName option ?arg arg ...?}
\]

*Option* and the *args* determine the exact behavior of the command. The following widget commands are possible for canvas widgets:

`pathName addtag` `tag` `searchSpec` `?arg arg ...?`

For each item that meets the constraints specified by `searchSpec` and the `args`, add `tag` to the list of tags associated with the item if it isn't already present on that list. It is possible that no items will satisfy the constraints given by `searchSpec` and `args`, in which case the command has no effect. This command returns an empty string as result. `SearchSpec` and `arg`s may take any of the following forms:

- **above** `tagOrId`
  Selects the item just after (above) the one given by `tagOrId` in the display list. If `tagOrId` denotes more than one item, then the last (topmost) of these items in the display list is used.

- **all**
  Selects all the items in the canvas.
below tagOrId
Selects the item just before (below) the one given by tagOrId in the display list. If tagOrId denotes more than one item, then the first (lowest) of these items in the display list is used.

closest x y ?halo? ?start?
Selects the item closest to the point given by x and y. If more than one item is at the same closest distance (e.g. two items overlap the point), then the top—most of these items (the last one in the display list) is used. If halo is specified, then it must be a non−negative value. Any item closer than halo to the point is considered to overlap it. The start argument may be used to step circularly through all the closest items. If start is specified, it names an item using a tag or id (if by tag, it selects the first item in the display list with the given tag). Instead of selecting the topmost closest item, this form will select the topmost closest item that is below start in the display list; if no such item exists, then the selection behaves as if the start argument had not been specified.

closed x1 y1 x2 y2
Selects all the items completely enclosed within the rectangular region given by x1, y1, x2, and y2. X1 must be no greater then x2 and y1 must be no greater than y2.

overlapping x1 y1 x2 y2
Selects all the items that overlap or are enclosed within the rectangular region given by x1, y1, x2, and y2. X1 must be no greater then x2 and y1 must be no greater than y2.

withtag tagOrId
Selects all the items given by tagOrId.

pathName bbox tagOrId ?tagOrId tagOrId ...?
Returns a list with four elements giving an approximate bounding box for all the items named by the tagOrId arguments. The list has the form ``x1 y1 x2 y2” such that the drawn areas of all the named elements are within the region bounded by x1 on the left, x2 on the right, y1 on the top, and y2 on the bottom. The return value may overestimate the actual bounding box by a few pixels. If no items match any of the tagOrId arguments or if the matching items have empty bounding boxes (i.e. they have nothing to display) then an empty string is returned.

pathName bind tagOrId ?sequence? ?command?
This command associates command with all the items given by tagOrId such that whenever the event sequence given by sequence occurs for one of the items the command will be invoked. This widget command is similar to the bind command except that it operates on items in a canvas rather than entire widgets. See the bind manual entry for complete details on the syntax of sequence and the substitutions performed on command before invoking it. If all arguments are specified then a new binding is created, replacing any existing binding for the same sequence and tagOrId (if the first character of command is ```+” then command augments an existing binding rather than replacing it). In this case the return value is an empty string. If command is omitted then the command returns the command associated with tagOrId and sequence (an error occurs if there is no such binding). If both command and sequence are omitted then the command returns a list of all the sequences for which bindings have been defined for tagOrId.

The only events for which bindings may be specified are those related to the mouse and keyboard (such as Enter, Leave, ButtonPress, Motion, and KeyPress) or virtual events. The handling of events in canvases uses the current item defined in ITEM IDS AND TAGS above. Enter and Leave events trigger for an item when it becomes the current item or ceases to be the current item; note that
these events are different than **Enter** and **Leave** events for windows. Mouse–related events are directed to the current item, if any. Keyboard–related events are directed to the focus item, if any (see the **focus** widget command below for more on this). If a virtual event is used in a binding, that binding can trigger only if the virtual event is defined by an underlying mouse–related or keyboard–related event.

It is possible for multiple bindings to match a particular event. This could occur, for example, if one binding is associated with the item's id and another is associated with one of the item's tags. When this occurs, all of the matching bindings are invoked. A binding associated with the **all** tag is invoked first, followed by one binding for each of the item's tags (in order), followed by a binding associated with the item's id. If there are multiple matching bindings for a single tag, then only the most specific binding is invoked. A **continue** command in a binding script terminates that script, and a **break** command terminates that script and skips any remaining scripts for the event, just as for the **bind** command.

If bindings have been created for a canvas window using the **bind** command, then they are invoked in addition to bindings created for the canvas's items using the **bind** widget command. The bindings for items will be invoked before any of the bindings for the window as a whole.

**pathName canvasx screenx ?gridspacing?**

Given a window x–coordinate in the canvas **screenx**, this command returns the canvas x–coordinate that is displayed at that location. If **gridspacing** is specified, then the canvas coordinate is rounded to the nearest multiple of **gridspacing** units.

**pathName canvasy screeny ?gridspacing?**

Given a window y–coordinate in the canvas **screeny** this command returns the canvas y–coordinate that is displayed at that location. If **gridspacing** is specified, then the canvas coordinate is rounded to the nearest multiple of **gridspacing** units.

**pathName cget option**

Returns the current value of the configuration option given by **option**. **Option** may have any of the values accepted by the **canvas** command.

**pathName configure ?option? ?value? ?option value ...?**

Query or modify the configuration options of the widget. If no **option** is specified, returns a list describing all of the available options for **pathName** (see **Tk_ConfigureInfo** for information on the format of this list). If **option** is specified with no **value**, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no **option** is specified). If one or more **option–value** pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. **Option** may have any of the values accepted by the **canvas** command.

**pathName coords tagOrId ?x0 y0 ...?**

**pathName coords tagOrId ?coordList?**

Query or modify the coordinates that define an item. If no coordinates are specified, this command returns a list whose elements are the coordinates of the item named by **tagOrId**. If coordinates are specified, then they replace the current coordinates for the named item. If **tagOrId** refers to multiple items, then the first one in the display list is used.

**pathName create type x y ?x y ...? ?option value ...?**

**pathName create type coordList ?option value ...?**
Create a new item in `pathName` of type `type`. The exact format of the arguments after `type` depends on `type`, but usually they consist of the coordinates for one or more points, followed by specifications for zero or more item options. See the subsections on individual item types below for more on the syntax of this command. This command returns the id for the new item.

`pathName dchars` `tagOrId` `first` ?`last`?
For each item given by `tagOrId`, delete the characters, or coordinates, in the range given by `first` and `last`, inclusive. If some of the items given by `tagOrId` don’t support indexing operations then they ignore `dchars`. Text items interpret `first` and `last` as indices to a character, line and polygon items interpret them indices to a coordinate (an x,y pair). Indices are described in INDICES above. If `last` is omitted, it defaults to `first`. This command returns an empty string.

`pathName delete` ?`tagOrId` `tagOrId` ...
Delete each of the items given by each `tagOrId`, and return an empty string.

`pathName dtag` `tagOrId` ?`tagToDelete`?
For each of the items given by `tagOrId`, delete the tag given by `tagToDelete` from the list of those associated with the item. If an item doesn’t have the tag `tagToDelete` then the item is unaffected by the command. If `tagToDelete` is omitted then it defaults to `tagOrId`. This command returns an empty string.

`pathName find` `searchCommand` ?`arg` `arg` ...
This command returns a list consisting of all the items that meet the constraints specified by `searchCommand` and `arg`'s. `SearchCommand` and `args` have any of the forms accepted by the `addtag` command. The items are returned in stacking order, with the lowest item first.

`pathName focus` ?`tagOrId`?
Set the keyboard focus for the canvas widget to the item given by `tagOrId`. If `tagOrId` refers to several items, then the focus is set to the first such item in the display list that supports the insertion cursor. If `tagOrId` doesn't refer to any items, or if none of them support the insertion cursor, then the focus isn't changed. If `tagOrId` is an empty string, then the focus item is reset so that no item has the focus. If `tagOrId` is not specified then the command returns the id for the item that currently has the focus, or an empty string if no item has the focus.

Once the focus has been set to an item, the item will display the insertion cursor and all keyboard events will be directed to that item. The focus item within a canvas and the focus window on the screen (set with the `focus` command) are totally independent: a given item doesn't actually have the input focus unless (a) its canvas is the focus window and (b) the item is the focus item within the canvas. In most cases it is advisable to follow the `focus` widget command with the `focus` command to set the focus window to the canvas (if it wasn't there already).

`pathName gettags` `tagOrId`
Return a list whose elements are the tags associated with the item given by `tagOrId`. If `tagOrId` refers to more than one item, then the tags are returned from the first such item in the display list. If `tagOrId` doesn't refer to any items, or if the item contains no tags, then an empty string is returned.

`pathName icursor` `tagOrId` `index`
Set the position of the insertion cursor for the item(s) given by `tagOrId` to just before the character whose position is given by `index`. If some or all of the items given by `tagOrId` don't support an insertion cursor then this command has no effect on them. See INDICES above for a description of the legal forms for `index`. Note: the insertion cursor is only displayed in an item if that item currently has the keyboard focus (see the widget command `focus`, below), but the cursor position may be set...
even when the item doesn't have the focus. This command returns an empty string.

`pathName index tagOrId index`

This command returns a decimal string giving the numerical index within `tagOrId` corresponding to `index`. Index gives a textual description of the desired position as described in INDICES above. Text items interpret `index` as an index to a character, line and polygon items interpret it as an index to a coordinate (an x,y pair). The return value is guaranteed to lie between 0 and the number of characters, or coordinates, within the item, inclusive. If `tagOrId` refers to multiple items, then the index is processed in the first of these items that supports indexing operations (in display list order).

`pathName insert tagOrId beforeThis string`

For each of the items given by `tagOrId`, if the item supports text or coordinate, insertion then `string` is inserted into the item's text just before the character, or coordinate, whose index is `beforeThis`. Text items interpret `beforeThis` as an index to a character, line and polygon items interpret it as an index to a coordinate (an x,y pair). For lines and polygons the `string` must be a valid coordinate sequence. See INDICES above for information about the forms allowed for `beforeThis`. This command returns an empty string.

`pathName itemcget tagOrId option`

Returns the current value of the configuration option for the item given by `tagOrId` whose name is `option`. This command is similar to the `cget` widget command except that it applies to a particular item rather than the widget as a whole. `Option` may have any of the values accepted by the `create` widget command when the item was created. If `tagOrId` is a tag that refers to more than one item, the first (lowest) such item is used.

`pathName itemconfigure tagOrId ?option? ?value? ?option value ...?`

This command is similar to the `configure` widget command except that it modifies item-specific options for the items given by `tagOrId` instead of modifying options for the overall canvas widget. If no `option` is specified, returns a list describing all of the available options for the first item given by `tagOrId` (see `Tk_ConfigureInfo` for information on the format of this list). If `option` is specified with no `value`, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no `option` is specified). If one or more `option-value` pairs are specified, then the command modifies the given widget option(s) to have the given value(s) in each of the items given by `tagOrId`; in this case the command returns an empty string. The `options` and `values` are the same as those permissible in the `create` widget command when the item(s) were created; see the sections describing individual item types below for details on the legal options.

`pathName lower tagOrId ?belowThis?`

Move all of the items given by `tagOrId` to a new position in the display list just before the item given by `belowThis`. If `tagOrId` refers to more than one item then all are moved but the relative order of the moved items will not be changed. `BelowThis` is a tag or id; if it refers to more than one item then the first (lowest) of these items in the display list is used as the destination location for the moved items. Note: this command has no effect on window items. Window items always obscure other item types, and the stacking order of window items is determined by the `raise` and `lower` commands, not the `raise` and `lower` widget commands for canvases. This command returns an empty string.

`pathName move tagOrId xAmount yAmount`

Move each of the items given by `tagOrId` in the canvas coordinate space by adding `xAmount` to the x-coordinate of each point associated with the item and `yAmount` to the y-coordinate of each point associated with the item. This command returns an empty string.
Generate a Postscript representation for part or all of the canvas. If the -file option is specified then the Postscript is written to a file and an empty string is returned; otherwise the Postscript is returned as the result of the command. If the interpreter that owns the canvas is marked as safe, the operation will fail because safe interpreters are not allowed to write files. If the -channel option is specified, the argument denotes the name of a channel already opened for writing. The Postscript is written to that channel, and the channel is left open for further writing at the end of the operation. The Postscript is created in Encapsulated Postscript form using version 3.0 of the Document Structuring Conventions. Note: by default Postscript is only generated for information that appears in the canvas's window on the screen. If the canvas is freshly created it may still have its initial size of 1x1 pixel so nothing will appear in the Postscript. To get around this problem either invoke the "update" command to wait for the canvas window to reach its final size, or else use the -width and -height options to specify the area of the canvas to print. The option=value argument pairs provide additional information to control the generation of Postscript. The following options are supported:

- colormap varName

VarName must be the name of an array variable that specifies a color mapping to use in the Postscript. Each element of varName must consist of Postscript code to set a particular color value (e.g. `1.0 1.0 0.0 setrgbcolor`). When outputting color information in the Postscript, Tk checks to see if there is an element of varName with the same name as the color. If so, Tk uses the value of the element as the Postscript command to set the color. If this option hasn't been specified, or if there isn't an entry in varName for a given color, then Tk uses the red, green, and blue intensities from the X color.

- colormode mode

Specifies how to output color information. Mode must be either color (for full color output), gray (convert all colors to their gray-scale equivalents) or mono (convert all colors to black or white).

- file fileName

Specifies the name of the file in which to write the Postscript. If this option isn't specified then the Postscript is returned as the result of the command instead of being written to a file.

- fontmap varName

VarName must be the name of an array variable that specifies a font mapping to use in the Postscript. Each element of varName must consist of a Tcl list with two elements, which are the name and point size of a Postscript font. When outputting Postscript commands for a particular font, Tk checks to see if varName contains an element with the same name as the font. If there is such an element, then the font information contained in that element is used in the Postscript. Otherwise Tk attempts to guess what Postscript font to use. Tk's guesses generally only work for well-known fonts such as Times and Helvetica and Courier, and only if the X font name does not omit any dashes up through the point size. For example, --*--Courier-Bold–R–Normal--*--120--* will work but *Courier–Bold–R–Normal*120* will not; Tk needs the dashes to parse the font name).

- height size

Specifies the height of the area of the canvas to print. Defaults to the height of the canvas window.

- pageanchor anchor
Specifies which point of the printed area of the canvas should appear over the positioning point on the page (which is given by the \(-pagex\) and \(-pagey\) options). For example, \(-pageanchor n\) means that the top center of the area of the canvas being printed (as it appears in the canvas window) should be over the positioning point. Defaults to center.

\(-pageheight\) \(size\)

Specifies that the Postscript should be scaled in both x and y so that the printed area is \(size\) high on the Postscript page. \(Size\) consists of a floating-point number followed by \(c\) for centimeters, \(i\) for inches, \(m\) for millimeters, or \(p\) or nothing for printer's points (1/72 inch). Defaults to the height of the printed area on the screen. If both \(-pageheight\) and \(-pagewidth\) are specified then the scale factor from \(-pagewidth\) is used (non-uniform scaling is not implemented).

\(-pagewidth\) \(size\)

Specifies that the Postscript should be scaled in both x and y so that the printed area is \(size\) wide on the Postscript page. \(Size\) has the same form as for \(-pageheight\). Defaults to the width of the printed area on the screen. If both \(-pageheight\) and \(-pagewidth\) are specified then the scale factor from \(-pagewidth\) is used (non-uniform scaling is not implemented).

\(-pagex\) \(position\)

\(Position\) gives the x-coordinate of the positioning point on the Postscript page, using any of the forms allowed for \(-pageheight\). Used in conjunction with the \(-pagey\) and \(-pageanchor\) options to determine where the printed area appears on the Postscript page. Defaults to the center of the page.

\(-pagey\) \(position\)

\(Position\) gives the y-coordinate of the positioning point on the Postscript page, using any of the forms allowed for \(-pageheight\). Used in conjunction with the \(-pagex\) and \(-pageanchor\) options to determine where the printed area appears on the Postscript page. Defaults to the center of the page.

\(-rotate\) \(boolean\)

\(Boolean\) specifies whether the printed area is to be rotated 90 degrees. In non-rotated output the x-axis of the printed area runs along the short dimension of the page ("portrait" orientation); in rotated output the x-axis runs along the long dimension of the page ("landscape" orientation). Defaults to non-rotated.

\(-width\) \(size\)

Specifies the width of the area of the canvas to print. Defaults to the width of the canvas window.

\(-x\) \(position\)

Specifies the x-coordinate of the left edge of the area of the canvas that is to be printed, in canvas coordinates, not window coordinates. Defaults to the coordinate of the left edge of the window.

\(-y\) \(position\)

Specifies the y-coordinate of the top edge of the area of the canvas that is to be printed, in canvas coordinates, not window coordinates. Defaults to the coordinate of the top edge of the window.

\(pathName\) \(raise\) \(tagOrId\) ?\(aboveThis\)?

Move all of the items given by \(tagOrId\) to a new position in the display list just after the item given by \(aboveThis\). If \(tagOrId\) refers to more than one item then all are moved but the relative order of the
moved items will not be changed. AboveThis is a tag or id; if it refers to more than one item then the last (topmost) of these items in the display list is used as the destination location for the moved items. Note: this command has no effect on window items. Window items always obscure other item types, and the stacking order of window items is determined by the raise and lower commands, not the raise and lower widget commands for canvases. This command returns an empty string.

`pathName scale tagOrId xOrigin yOrigin xScale yScale`

Rescale all of the items given by `tagOrId` in canvas coordinate space. `xOrigin` and `yOrigin` identify the origin for the scaling operation and `xScale` and `yScale` identify the scale factors for x− and y−coordinates, respectively (a scale factor of 1.0 implies no change to that coordinate). For each of the points defining each item, the x−coordinate is adjusted to change the distance from `xOrigin` by a factor of `xScale`. Similarly, each y−coordinate is adjusted to change the distance from `yOrigin` by a factor of `yScale`. This command returns an empty string.

`pathName scan option args`

This command is used to implement scanning on canvases. It has two forms, depending on `option`:

`pathName scan mark x y`

Records `x` and `y` and the canvas's current view; used in conjunction with later scan dragto commands. Typically this command is associated with a mouse button press in the widget and `x` and `y` are the coordinates of the mouse. It returns an empty string.

`pathName scan dragto x y ?gain?`

This command computes the difference between its `x` and `y` arguments (which are typically mouse coordinates) and the `x` and `y` arguments to the last scan mark command for the widget. It then adjusts the view by `gain` times the difference in coordinates, where `gain` defaults to 10. This command is typically associated with mouse motion events in the widget, to produce the effect of dragging the canvas at high speed through its window. The return value is an empty string.

`pathName select option ?tagOrId arg?`

Manipulates the selection in one of several ways, depending on `option`. The command may take any of the forms described below. In all of the descriptions below, `tagOrId` must refer to an item that supports indexing and selection; if it refers to multiple items then the first of these that supports indexing and the selection is used. `Index` gives a textual description of a position within `tagOrId`, as described in INDICES above.

`pathName select adjust tagOrId index`

Locate the end of the selection in `tagOrId` nearest to the character given by `index`, and adjust that end of the selection to be at `index` (i.e. including but not going beyond `index`). The other end of the selection is made the anchor point for future select to commands. If the selection isn't currently in `tagOrId` then this command behaves the same as the select to widget command. Returns an empty string.

`pathName select clear`

Clear the selection if it is in this widget. If the selection isn't in this widget then the command has no effect. Returns an empty string.

`pathName select from tagOrId index`

Set the selection anchor point for the widget to be just before the character given by `index` in the item given by `tagOrId`. This command doesn't change the selection; it just sets the fixed
end of the selection for future select to commands. Returns an empty string.

**pathName select item**
Returns the id of the selected item, if the selection is in an item in this canvas. If the selection is not in this canvas then an empty string is returned.

**pathName select to tagOrId index**
Set the selection to consist of those characters of tagOrId between the selection anchor point and index. The new selection will include the character given by index; it will include the character given by the anchor point only if index is greater than or equal to the anchor point. The anchor point is determined by the most recent select adjust or select from command for this widget. If the selection anchor point for the widget isn't currently in tagOrId, then it is set to the same character given by index. Returns an empty string.

**pathName type tagOrId**
Returns the type of the item given by tagOrId, such as rectangle or text. If tagOrId refers to more than one item, then the type of the first item in the display list is returned. If tagOrId doesn't refer to any items at all then an empty string is returned.

**pathName xview ?args?**
This command is used to query and change the horizontal position of the information displayed in the canvas's window. It can take any of the following forms:

**pathName xview**
Returns a list containing two elements. Each element is a real fraction between 0 and 1; together they describe the horizontal span that is visible in the window. For example, if the first element is .2 and the second element is .6, 20% of the canvas's area (as defined by the−scrollregion option) is off−screen to the left, the middle 40% is visible in the window, and 40% of the canvas is off−screen to the right. These are the same values passed to scrollbars via the−xscrollcommand option.

**pathName xview moveto fraction**
Adjusts the view in the window so that fraction of the total width of the canvas is off−screen to the left. Fraction must be a fraction between 0 and 1.

**pathName xview scroll number what**
This command shifts the view in the window left or right according to number and what. Number must be an integer. What must be either units or pages or an abbreviation of one of these. If what is units, the view adjusts left or right in units of the xScrollIncrement option, if it is greater than zero, or in units of one−tenth the window's width otherwise. If what is pages then the view adjusts in units of nine−tenths the window's width. If number is negative then information farther to the left becomes visible; if it is positive then information farther to the right becomes visible.

**pathName yview ?args?**
This command is used to query and change the vertical position of the information displayed in the canvas's window. It can take any of the following forms:

**pathName yview**
Returns a list containing two elements. Each element is a real fraction between 0 and 1; together they describe the vertical span that is visible in the window. For example, if the first element is .6 and the second element is 1.0, the lowest 40% of the canvas's area (as defined by
the \texttt{scrollregion} option is visible in the window. These are the same values passed to
scrollbars via the \texttt{yscrollcommand} option.

\begin{verbatim}
pathName yview moveto fraction
\end{verbatim}

Adjusts the view in the window so that \textit{fraction} of the canvas's area is off–screen to the top. \textit{Fraction} is a fraction between 0 and 1.

\begin{verbatim}
pathName yview scroll number what
\end{verbatim}

This command adjusts the view in the window up or down according to \textit{number} and \textit{what}. \textit{Number} must be an integer. \textit{What} must be either \texttt{units} or \texttt{pages}. If \textit{what} is \texttt{units}, the view adjusts up or down in units of the \texttt{yScrollIncrement} option, if it is greater than zero, or in units of one–tenth the window's height otherwise. If \textit{what} is \texttt{pages} then the view adjusts in units of nine–tenths the window's height. If \textit{number} is negative then higher information becomes visible; if it is positive then lower information becomes visible.

\section*{Overview of Item Types}

The sections below describe the various types of items supported by canvas widgets. Each item type is characterized by two things: first, the form of the \texttt{create} command used to create instances of the type; and second, a set of configuration options for items of that type, which may be used in the \texttt{create} and \texttt{itemconfigure} widget commands. Most items don't support indexing or selection or the commands related to them, such as \texttt{index} and \texttt{insert}. Where items do support these facilities, it is noted explicitly in the descriptions below. At present, text, line and polygon items provide this support. For lines and polygons the indexing facility is used to manipulate the coordinates of the item.

\section*{Common Item Options}

Many items share a common set of options. These options are explained here, and then referred to be each widget type for brevity.

\begin{verbatim}
--dash pattern
--activedash pattern
--disableddash pattern
\end{verbatim}

This option specifies dash patterns for the normal, active state, and disabled state of an item. \textit{pattern} may have any of the forms accepted by \texttt{Tk.GetDash}. If the dash options are omitted then the default is a solid outline. See "DASH PATTERNS" for more information.

\begin{verbatim}
--dashoffset offset
\end{verbatim}

The starting \textit{offset} in pixels into the pattern provided by the \texttt{--dash} option. \texttt{--dashoffset} is ignored if there is no \texttt{--dash} pattern. The \textit{offset} may have any of the forms described in the \texttt{COORDINATES} section above.

\begin{verbatim}
--fill color
--activefill color
--disabledfill color
\end{verbatim}

Specifies the color to be used to fill item's area. in its normal, active, and disabled states, \textit{Color} may have any of the forms accepted by \texttt{Tk.GetColor}. If \textit{color} is an empty string (the default), then the item will not be filled. For the line item, it specifies the color of the line drawn. For the text item, it specifies the foreground color of the text.
This option specifies the color that should be used to draw the outline of the item in its normal, active and disabled states. Color may have any of the forms accepted by `Tk_GetColor`. This option defaults to black. If `color` is specified as an empty string then no outline is drawn for the item.

Specifies the offset of stipples. The offset value can be of the form x,y or side, where side can be n, ne, e, se, s, sw, w, nw, or center. In the first case the origin is the origin of the toplevel of the current window. For the canvas itself and canvas objects the origin is the canvas origin, but putting # in front of the coordinate pair indicates using the toplevel origin instead. For canvas objects, the `-offset` option is used for stippling as well. For the line and polygon canvas items you can also specify an index as argument, which connects the stipple origin to one of the coordinate points of the line/polygon.

This option specifies stipple patterns that should be used to draw the outline of the item in its normal, active and disabled states. Indicates that the outline for the item should be drawn with a stipple pattern; `bitmap` specifies the stipple pattern to use, in any of the forms accepted by `Tk_GetBitmap`. If the `-outline` option hasn't been specified then this option has no effect. If `bitmap` is an empty string (the default), then the outline is drawn in a solid fashion.

This option specifies stipple patterns that should be used to fill the item in its normal, active and disabled states. `bitmap` specifies the stipple pattern to use, in any of the forms accepted by `Tk_GetBitmap`. If the `-fill` option hasn't been specified then this option has no effect. If `bitmap` is an empty string (the default), then filling is done in a solid fashion. For the text item, it affects the actual text.

This allows an item to override the canvas widget's global `state` option. It takes the same values: normal, disabled or hidden.

Specifies a set of tags to apply to the item. `TagList` consists of a list of tag names, which replace any existing tags for the item. `TagList` may be an empty list.

Specifies the width of the outline to be drawn around the item's region, in its normal, active and disabled states. `outlineWidth` may be in any of the forms described in the `COORDINATES` section above. If the `-outline` option has been specified as an empty string then this option has no effect. This option defaults to 1.0. For arcs, wide outlines will be drawn centered on the edges of the arc's region.
ARC ITEMS

Items of type arc appear on the display as arc-shaped regions. An arc is a section of an oval delimited by two angles (specified by the –start and –extent options) and displayed in one of several ways (specified by the –style option). Arcs are created with widget commands of the following form:

```
pathName create arc x1 y1 x2 y2 ?option value option value ...?
pathName create arc coordList ?option value option value ...?
```

The arguments $x1$, $y1$, $x2$, and $y2$ or coordList give the coordinates of two diagonally opposite corners of a rectangular region enclosing the oval that defines the arc. After the coordinates there may be any number of option–value pairs, each of which sets one of the configuration options for the item. These same option–value pairs may be used in itemconfigure widget commands to change the item's configuration.

The following standard options are supported by arcs:

- dash
- activedash
- disabledash
- dashoffset
- fill
- activefill
- disabledfill
- offset
- outline
- activeoutline
- disabledoutline
- linestipple
- activeoutlinestipple
- disabledoutlinestipple
- stipple
- activestipple
- disabledstipple
- state
- tags
- width
- activewidth
- disabledwidth

The following extra options are supported for arcs:

- **–extent degrees**
  Specifies the size of the angular range occupied by the arc. The arc’s range extends for degrees degrees counter-clockwise from the starting angle given by the –start option. Degrees may be negative. If it is greater than 360 or less than –360, then degrees modulo 360 is used as the extent.

- **–start degrees**
  Specifies the beginning of the angular range occupied by the arc. Degrees is given in units of degrees measured counter-clockwise from the 3–o’clock position; it may be either positive or negative.

- **–style type**
  Specifies how to draw the arc. If type is pieslice (the default) then the arc's region is defined by a
section of the oval's perimeter plus two line segments, one between the center of the oval and each end of the perimeter section. If type is chord then the arc's region is defined by a section of the oval's perimeter plus a single line segment connecting the two end points of the perimeter section. If type is arc then the arc's region consists of a section of the perimeter alone. In this last case the –fill option is ignored.

BITMAP ITEMS

Items of type bitmap appear on the display as images with two colors, foreground and background. Bitmaps are created with widget commands of the following form:

```
pathName create bitmap x y ?option value option value ...?
pPathName create bitmap coordList ?option value option value ...?
```

The arguments x and y or coordList specify the coordinates of a point used to position the bitmap on the display (see the –anchor option below for more information on how bitmaps are displayed). After the coordinates there may be any number of option–value pairs, each of which sets one of the configuration options for the item. These same option–value pairs may be used in itemconfigure widget commands to change the item's configuration.

The following standard options are supported by bitmaps:

- -state
- -tags

The following extra options are supported for bitmaps:

- **anchor** anchorPos
  
  AnchorPos tells how to position the bitmap relative to the positioning point for the item; it may have any of the forms accepted by Tk_GetAnchor. For example, if anchorPos is center then the bitmap is centered on the point; if anchorPos is n then the bitmap will be drawn so that its top center point is at the positioning point. This option defaults to center.

- **background** color
- **activebackground** bitmap
- **disabledbackground** bitmap
  
  Specifies the color to use for each of the bitmap's '0' valued pixels in its normal, active and disabled states. Color may have any of the forms accepted by Tk_GetColor. If this option isn't specified, or if it is specified as an empty string, then nothing is displayed where the bitmap pixels are 0; this produces a transparent effect.

- **bitmap** bitmap
- **activebitmap** bitmap
- **disabledbitmap** bitmap
  
  Specifies the bitmaps to display in the item in its normal, active and disabled states. Bitmap may have any of the forms accepted by Tk_GetBitmap.

- **foreground** color
- **activeforeground** bitmap
- **disabledforeground** bitmap
Specifies the color to use for each of the bitmap's '1' valued pixels in its normal, active and disabled states. Color may have any of the forms accepted by `Tk_GetColor` and defaults to `black`.

**IMAGE ITEMS**

Items of type `image` are used to display images on a canvas. Images are created with widget commands of the following form:

```tcl
pathName create image x y ?option value option value ...?
pathName create image coordList ?option value option value ...?
```

The arguments `x` and `y` or `coordList` specify the coordinates of a point used to position the image on the display (see the `-anchor` option below for more information). After the coordinates there may be any number of `option–value` pairs, each of which sets one of the configuration options for the item. These same `option–value` pairs may be used in `itemconfigure` widget commands to change the item's configuration.

The following standard options are supported by images:

- `-state`
- `-tags`

The following extra options are supported for images:

- `-anchor anchorPos`
  AnchorPos tells how to position the image relative to the positioning point for the item; it may have any of the forms accepted by `Tk_GetAnchor`. For example, if `anchorPos` is `center` then the image is centered on the point; if `anchorPos` is `n` then the image will be drawn so that its top center point is at the positioning point. This option defaults to `center`.

- `-image name`
- `-activeimage name`
- `-disabledimage name`
  Specifies the name of the images to display in the item in is normal, active and disabled states. This image must have been created previously with the `image create` command.

**LINE ITEMS**

Items of type `line` appear on the display as one or more connected line segments or curves. Line items support coordinate indexing operations using the canvas widget commands: `dchars, index, insert`. Lines are created with widget commands of the following form:

```tcl
pathName create line x1 y1... xn yn ?option value option value ...?
pathName create line coordList ?option value option value ...?
```

The arguments `x1` through `yn` or `coordList` give the coordinates for a series of two or more points that describe a series of connected line segments. After the coordinates there may be any number of `option–value` pairs, each of which sets one of the configuration options for the item. These same `option–value` pairs may be used in `itemconfigure` widget commands to change the item's configuration.
The following standard options are supported by lines:

- `dash`
- `activedash`
- `disableddash`
- `dashoffset`
- `fill`
- `activefill`
- `disabledfill`
- `stipple`
- `activestipple`
- `disabledstipple`
- `state`
- `tags`
- `width`
- `activewidth`
- `disabledwidth`

The following extra options are supported for lines:

--arrow where
  Indicates whether or not arrowheads are to be drawn at one or both ends of the line. Where must have one of the values none (for no arrowheads), first (for an arrowhead at the first point of the line), last (for an arrowhead at the last point of the line), or both (for arrowheads at both ends). This option defaults to none.

--arrowshape shape
  This option indicates how to draw arrowheads. The shape argument must be a list with three elements, each specifying a distance in any of the forms described in the COORDINATES section above. The first element of the list gives the distance along the line from the neck of the arrowhead to its tip. The second element gives the distance along the line from the trailing points of the arrowhead to the tip, and the third element gives the distance from the outside edge of the line to the trailing points. If this option isn't specified then Tk picks a ``reasonable`` shape.

--capstyle style
  Specifies the ways in which caps are to be drawn at the endpoints of the line. Style may have any of the forms accepted by Tk_GetCapStyle (butt, projecting, or round). If this option isn't specified then it defaults to butt. Where arrowheads are drawn the cap style is ignored.

--joinstyle style
  Specifies the ways in which joints are to be drawn at the vertices of the line. Style may have any of the forms accepted by Tk_GetCapStyle (bevel, miter, or round). If this option isn't specified then it defaults to miter. If the line only contains two points then this option is irrelevant.

--smooth smoothMethod
  smoothMethod must have one of the forms accepted by Tk_GetBoolean or a line smoothing method. Only bezier is supported in the core, but more can be added at runtime. If a boolean false value or empty string is given, no smoothing is applied. A boolean truth value assume bezier smoothing. It indicates whether or not the line should be drawn as a curve. If so, the line is rendered as a set of parabolic splines: one spline is drawn for the first and second line segments, one for the second and third, and so on. Straight–line segments can be generated within a curve by duplicating the end–points of the desired line segment.
−splinesteps number
  Specifies the degree of smoothness desired for curves: each spline will be approximated with number line segments. This option is ignored unless the −smooth option is true.

OVAL ITEMS

Items of type oval appear as circular or oval regions on the display. Each oval may have an outline, a fill, or both. Ovals are created with widget commands of the following form:

    pathName create oval x1 y1 x2 y2 ?option value option value ...?
    pathName create oval coordList ?option value option value ...?

The arguments x1, y1, x2, and y2 or coordList give the coordinates of two diagonally opposite corners of a rectangular region enclosing the oval. The oval will include the top and left edges of the rectangle not the lower or right edges. If the region is square then the resulting oval is circular; otherwise it is elongated in shape. After the coordinates there may be any number of option−value pairs, each of which sets one of the configuration options for the item. These same option−value pairs may be used in itemconfigure widget commands to change the item's configuration.

The following standard options are supported by ovals:

−dash
−activedash
−disableddash
−dashoffset
−fill
−activefill
−disabledfill
−offset
−outline
−activeoutline
−disabledoutline
−outlinestipple
−activeoutlinestipple
−disabledoutlinestipple
−stipple
−activestipple
−disabledstipple
−state
−tags
−width
−activewidth
−disabledwidth

POLYGON ITEMS

Items of type polygon appear as polygonal or curved filled regions on the display. Polygon items support coordinate indexing operations using the canvas widget commands: dchars, index, insert. Polygons are created with widget commands of the following form:
pathName create polygon x1 y1 ... xn yn ?option value option value ...?
pathName create polygon coordList ?option value option value ...?

The arguments x1 through yn or coordList specify the coordinates for three or more points that define a polygon. The first point should not be repeated as the last to close the shape; Tk will automatically close the periphery between the first and last points. After the coordinates there may be any number of option–value pairs, each of which sets one of the configuration options for the item. These same option–value pairs may be used in itemconfigure widget commands to change the item’s configuration.

The following standard options are supported by polygons:

- dash
- activedash
- disabledash
- dashoffset
- fill
- activefill
- disabledfill
- offset
- outline
- activeoutline
- disabledoutline
- outlinestipple
- activeoutlinestipple
- disabledoutlinestipple
- stipple
- activestipple
- disabledstipple
- state
- tags
- width
- activewidth
- disabledwidth

The following extra options are supported for polygons:

-joinstyle style
  Specifies the ways in which joints are to be drawn at the vertices of the outline. Style may have any of the forms accepted by Tk_GetCapStyle (bevel, miter, or round). If this option isn’t specified then it defaults to miter.

-smooth boolean
  Boolean must have one of the forms accepted by Tk_GetBoolean. It indicates whether or not the polygon should be drawn with a curved perimeter. If so, the outline of the polygon becomes a set of parabolic splines, one spline for the first and second line segments, one for the second and third, and so on. Straight–line segments can be generated in a smoothed polygon by duplicating the end–points of the desired line segment.

-splinesteps number
  Specifies the degree of smoothness desired for curves: each spline will be approximated with number line segments. This option is ignored unless the –smooth option is true.
Polygon items are different from other items such as rectangles, ovals and arcs in that interior points are considered to be "inside" a polygon (e.g. for purposes of the find closest and find overlapping widget commands) even if it is not filled. For most other item types, an interior point is considered to be inside the item only if the item is filled or if it has neither a fill nor an outline. If you would like an unfilled polygon whose interior points are not considered to be inside the polygon, use a line item instead.

**RECTANGLE ITEMS**

Items of type rectangle appear as rectangular regions on the display. Each rectangle may have an outline, a fill, or both. Rectangles are created with widget commands of the following form:

```
pathName create rectangle x1 y1 x2 y2 ?option value option value ...?
pathName create rectangle coordList ?option value option value ...?
```

The arguments $x1$, $y1$, $x2$, and $y2$ or coordList give the coordinates of two diagonally opposite corners of the rectangle (the rectangle will include its upper and left edges but not its lower or right edges). After the coordinates there may be any number of option–value pairs, each of which sets one of the configuration options for the item. These same option–value pairs may be used in itemconfigure widget commands to change the item's configuration.

The following standard options are supported by rectangles:

- dash
- activedash
- disabledash
- dashoffset
- fill
- activefill
- disabledfill
- offset
- outline
- activeoutline
- disabledoutline
- outlinestipple
- activeoutlinestipple
- disabledoutlinestipple
- stipple
- activestipple
- disabledstipple
- state
- tags
- width
- activewidth
- disabledwidth

**TEXT ITEMS**

A text item displays a string of characters on the screen in one or more lines. Text items support indexing and selection, along with the following text–related canvas widget commands: dchars, focus, icursor, index, insert, select. Text items are created with widget commands of the following form:
The arguments $x$ and $y$ or $coordList$ specify the coordinates of a point used to position the text on the display (see the options below for more information on how text is displayed). After the coordinates there may be any number of option-value pairs, each of which sets one of the configuration options for the item. These same option-value pairs may be used in `itemconfigure` widget commands to change the item's configuration.

The following standard options are supported by text items:

- `-fill`
- `-activefill`
- `-disabledfill`
- `-stipple`
- `-activestipple`
- `-disabledstipple`
- `-state`
- `-tags`

The following extra options are supported for text items:

- `-anchor anchorPos`
  
  AnchorPos tells how to position the text relative to the positioning point for the text; it may have any of the forms accepted by `Tk_GetAnchor`. For example, if `anchorPos` is `center` then the text is centered on the point; if `anchorPos` is `n` then the text will be drawn such that the top center point of the rectangular region occupied by the text will be at the positioning point. This option defaults to `center`.

- `-font fontName`
  
  Specifies the font to use for the text item. `fontName` may be any string acceptable to `Tk_GetFont`. If this option isn't specified, it defaults to a system-dependent font.

- `-justify how`
  
  Specifies how to justify the text within its bounding region. `how` must be one of the values `left`, `right`, or `center`. This option will only matter if the text is displayed as multiple lines. If the option is omitted, it defaults to `left`.

- `-text string`
  
  `String` specifies the characters to be displayed in the text item. Newline characters cause line breaks. The characters in the item may also be changed with the `insert` and `delete` widget commands. This option defaults to an empty string.

- `-width lineLength`
  
  Specifies a maximum line length for the text, in any of the forms described in the `COORDINATES` section above. If this option is zero (the default) the text is broken into lines only at newline characters. However, if this option is non-zero then any line that would be longer than `lineLength` is broken just before a space character to make the line shorter than `lineLength`; the space character is treated as if it were a newline character.
**WINDOW ITEMS**

Items of type `window` cause a particular window to be displayed at a given position on the canvas. Window items are created with widget commands of the following form:

```
pathName create window x y ?option value option value ...?
pathName create window coordList ?option value option value ...?
```

The arguments `x` and `y` or `coordList` specify the coordinates of a point used to position the window on the display (see the `−anchor` option below for more information on how bitmaps are displayed). After the coordinates there may be any number of `option−value` pairs, each of which sets one of the configuration options for the item. These same `option−value` pairs may be used in `itemconfigure` widget commands to change the item's configuration.

The following standard options are supported by window items:

−`−state`
−`−tags`

The following extra options are supported for window items:

−`−anchor anchorPos`

  AnchorPos tells how to position the window relative to the positioning point for the item; it may have any of the forms accepted by `Tk_GetAnchor`. For example, if `anchorPos` is `center` then the window is centered on the point; if `anchorPos` is `n` then the window will be drawn so that its top center point is at the positioning point. This option defaults to `center`.

−`−height pixels`

  Specifies the height to assign to the item's window. **Pixels** may have any of the forms described in the `COORDINATES` section above. If this option isn't specified, or if it is specified as an empty string, then the window is given whatever height it requests internally.

−`−width pixels`

  Specifies the width to assign to the item's window. **Pixels** may have any of the forms described in the `COORDINATES` section above. If this option isn't specified, or if it is specified as an empty string, then the window is given whatever width it requests internally.

−`−window pathName`

  Specifies the window to associate with this item. The window specified by `pathName` must either be a child of the canvas widget or a child of some ancestor of the canvas widget. **PathName** may not refer to a top−level window.

Note: due to restrictions in the ways that windows are managed, it is not possible to draw other graphical items (such as lines and images) on top of window items. A window item always obscures any graphics that overlap it, regardless of their order in the display list.

**APPLICATION−DEFINED ITEM TYPES**

It is possible for individual applications to define new item types for canvas widgets using C code. See the documentation for `Tk_CreateItemType`.

---

**Tcl8.4/Tk8.4 Manual**

**WINDOW ITEMS**

412
BINDINGS

In the current implementation, new canvases are not given any default behavior: you'll have to execute explicit Tcl commands to give the canvas its behavior.

CREDITS

Tk's canvas widget is a blatant ripoff of ideas from Joel Bartlett's ezd program. Ezd provides structured graphics in a Scheme environment and preceded canvases by a year or two. Its simple mechanisms for placing and animating graphical objects inspired the functions of canvases.

SEE ALSO

bind, font, image, scrollbar

KEYWORDS

canvas, widget

Copyright ' 1992−1994 The Regents of the University of California.
Copyright ' 1994−1996 Sun Microsystems, Inc.
Copyright ' 1997−1999 Scriptics Corporation.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
checkbutton

NAME

checkbutton – Create and manipulate checkbutton widgets

SYNOPSIS

checkbutton pathName ?options?

STANDARD OPTIONS

-activebackground, activeBackground, Foreground
-activelowgroung, activeForeground, Background
-anchor, anchor, Anchor
-background or -bg, background, Background
-bimap, bitmap, Bitmap
-borderwidth or -bd, borderWidth, BorderWidth
-compound, compound, Compound
-cursor, cursor, Cursor
-disabledforeground, disabledForeground, DisabledForeground
-font, font, Font
-foregroung or -fg, foreground, Foreground
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-image, image, Image
-justify, justify, Justify
-padx, padX, Pad
-pady, padY, Pad
-relief, relief, Relief
-takefocus, takeFocus, TakeFocus
-text, text, Text
-textvariable, textVariable, Variable
-underline, underline, Underline
-wraplength, wrapLength, WrapLength

WIDGET–SPECIFIC OPTIONS

-command, command, Command
-height, height, Height
-indicatoron, indicatorOn, IndicatorOn
-offrelief, offRelief, OffRelief
-offvalue, offValue, Value
-onvalue, onValue, Value
-overrelief, overRelief, OverRelief
-selectcolor, selectColor, Background
-selectimage, selectImage, SelectImage
-state, state, State
-variable, variable, Variable
DESCRIPTION

WIDGET COMMAND

- `pathName cget option`
- `pathName configure ?option? ?value option value ...`?
- `pathName deselect`
- `pathName flash`
- `pathName invoke`
- `pathName select`
- `pathName toggle`

BINDINGS

SEE ALSO

KEYWORDS

NAME

checkbutton – Create and manipulate checkbutton widgets

SYNOPSIS

`checkbutton pathName ?options?`

STANDARD OPTIONS

- `-activebackground, activeBackground, Foreground`
- `-activeforeground, activeForeground, Background`
- `-anchor, anchor, Anchor`
- `-background or -bg, background, Background`
- `-bitmap, bitmap, Bitmap`
- `-borderwidth or -bd, borderWidth, BorderWidth`
- `-compound, compound, Compound`
- `-cursor, cursor, Cursor`
- `-disabledforeground, disabledForeground, DisabledForeground`
- `-font, font, Font`
- `-foreground or -fg, foreground, Foreground`
- `-highlightbackground, highlightBackground, HighlightBackground`
- `-highlightcolor, highlightColor, HighlightColor`
- `-highlightthickness, highlightThickness, HighlightThickness`
- `-image, image, Image`
- `-justify, justify, Justify`
- `-padx, padX, Pad`
- `-pady, padY, Pad`
- `-relief, relief, Relief`
- `-takefocus, takeFocus, TakeFocus`
- `-text, text, Text`
–textvariable, textVariable, Variable
–underline, underline, Underline
–wraplength, wrapLength, WrapLength

WIDGET–SPECIFIC OPTIONS

Command–Line Name: –command
Database Name: command
Database Class: Command

Specifies a Tcl command to associate with the button. This command is typically invoked when mouse button 1 is released over the button window. The button's global variable (–variable option) will be updated before the command is invoked.

Command–Line Name: –height
Database Name: height
Database Class: Height

Specifies a desired height for the button. If an image or bitmap is being displayed in the button then the value is in screen units (i.e. any of the forms acceptable to Tk::GetPixels); for text it is in lines of text. If this option isn't specified, the button's desired height is computed from the size of the image or bitmap or text being displayed in it.

Command–Line Name: –indicatoron
Database Name: indicatorOn
Database Class: IndicatorOn

Specifies whether or not the indicator should be drawn. Must be a proper boolean value. If false, the relief option is ignored and the widget's relief is always sunken if the widget is selected and raised otherwise.

Command–Line Name: –offrelief
Database Name: offRelief
Database Class: OffRelief

Specifies the relief for the checkbutton when the indicator is not drawn and the checkbutton is off. The default value is "raised". By setting this option to "flat" and setting –indicatoron to false and –overrelief to raised, the effect is achieved of having a flat button that raises on mouse−over and which is depressed when activated. This is the behavior typically exhibited by the Bold, Italic, and Underline checkbuttons on the toolbar of a word−processor, for example.

Command–Line Name: –offvalue
Database Name: offValue
Database Class: Value

Specifies value to store in the button's associated variable whenever this button is deselected. Defaults to "0".

Command–Line Name: –onvalue
Database Name: onValue
Database Class: Value

Specifies value to store in the button's associated variable whenever this button is selected. Defaults to "1".

Command–Line Name: –overrelief
Database Name: overRelief
Database Class: **OverRelief**

Specifies an alternative relief for the checkbutton, to be used when the mouse cursor is over the widget. This option can be used to make toolbar buttons, by configuring `−relief flat −overrelief raised`. If the value of this option is the empty string, then no alternative relief is used when the mouse cursor is over the checkbutton. The empty string is the default value.

**Command−Line Name:** `−selectcolor`
**Database Name:** `selectColor`

**Database Class:** **Background**

Specifies a background color to use when the button is selected. If `indicatorOn` is true then the color applies to the indicator. Under Windows, this color is used as the background for the indicator regardless of the select state. If `indicatorOn` is false, this color is used as the background for the entire widget, in place of `background` or `activeBackground`, whenever the widget is selected. If specified as an empty string then no special color is used for displaying when the widget is selected.

**Command−Line Name:** `−selectimage`
**Database Name:** `selectImage`

**Database Class:** **SelectImage**

Specifies an image to display (in place of the `image` option) when the checkbutton is selected. This option is ignored unless the `image` option has been specified.

**Command−Line Name:** `−state`
**Database Name:** `state`

**Database Class:** **State**

Specifies one of three states for the checkbutton: `normal`, `active`, or `disabled`. In normal state the checkbutton is displayed using the `foreground` and `background` options. The active state is typically used when the pointer is over the checkbutton. In active state the checkbutton is displayed using the `activeForeground` and `activeBackground` options. Disabled state means that the checkbutton should be insensitive: the default bindings will refuse to activate the widget and will ignore mouse button presses. In this state the `disabledForeground` and `background` options determine how the checkbutton is displayed.

**Command−Line Name:** `−variable`
**Database Name:** `variable`

**Database Class:** **Variable**

Specifies name of global variable to set to indicate whether or not this button is selected. Defaults to the name of the button within its parent (i.e. the last element of the button window’s path name).

**Command−Line Name:** `−width`
**Database Name:** `width`

**Database Class:** **Width**

Specifies a desired width for the button. If an image or bitmap is being displayed in the button then the value is in screen units (i.e. any of the forms acceptable to `Tk_GetPixels`); for text it is in characters. If this option isn't specified, the button's desired width is computed from the size of the image or bitmap or text being displayed in it.

**DESCRIPTION**

The `checkbutton` command creates a new window (given by the `pathName` argument) and makes it into a checkbutton widget. Additional options, described above, may be specified on the command line or in the
option database to configure aspects of the checkbutton such as its colors, font, text, and initial relief. The `checkbutton` command returns its `pathName` argument. At the time this command is invoked, there must not exist a window named `pathName`, but `pathName`'s parent must exist.

A checkbutton is a widget that displays a textual string, bitmap or image and a square called an indicator. If text is displayed, it must all be in a single font, but it can occupy multiple lines on the screen (if it contains newlines or if wrapping occurs because of the `wrapLength` option) and one of the characters may optionally be underlined using the `underline` option. A checkbutton has all of the behavior of a simple button, including the following: it can display itself in either of three different ways, according to the `state` option; it can be made to appear raised, sunken, or flat; it can be made to flash; and it invokes a Tcl command whenever mouse button 1 is clicked over the checkbutton.

In addition, checkbuttons can be selected. If a checkbutton is selected then the indicator is normally drawn with a selected appearance, and a Tcl variable associated with the checkbutton is set to a particular value (normally 1). Under Unix, the indicator is drawn with a sunken relief and a special color. Under Windows, the indicator is drawn with a check mark inside. If the checkbutton is not selected, then the indicator is drawn with a deselected appearance, and the associated variable is set to a different value (typically 0). Under Unix, the indicator is drawn with a raised relief and no special color. Under Windows, the indicator is drawn without a check mark inside. By default, the name of the variable associated with a checkbutton is the same as the name used to create the checkbutton. The variable name, and the `on` and `off` values stored in it, may be modified with options on the command line or in the option database. Configuration options may also be used to modify the way the indicator is displayed (or whether it is displayed at all). By default a checkbutton is configured to select and deselect itself on alternate button clicks. In addition, each checkbutton monitors its associated variable and automatically selects and deselects itself when the variables value changes to and from the button's `on` value.

**WIDGET COMMAND**

The `checkbutton` command creates a new Tcl command whose name is `pathName`. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

`Option` and the `args` determine the exact behavior of the command. The following commands are possible for checkbutton widgets:

```
pathName cget option
```

Returns the current value of the configuration option given by `option`. `Option` may have any of the values accepted by the `checkbutton` command.

```
pathName configure ?option? ?value option value ...?
```

Query or modify the configuration options of the widget. If no `option` is specified, returns a list describing all of the available options for `pathName` (see `Tk_ConfigureInfo` for information on the format of this list). If `option` is specified with no `value`, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no `option` is specified). If one or more `option=value` pairs are specified, then the command modifies the
given widget option(s) to have the given value(s); in this case the command returns an empty string. Option may have any of the values accepted by the checkbutton command.

**pathName deselect**

Deselects the checkbutton and sets the associated variable to its ``off'' value.

**pathName flash**

Flashes the checkbutton. This is accomplished by redisplaying the checkbutton several times, alternating between active and normal colors. At the end of the flash the checkbutton is left in the same normal/active state as when the command was invoked. This command is ignored if the checkbutton's state is **disabled**.

**pathName invoke**

Does just what would have happened if the user invoked the checkbutton with the mouse: toggle the selection state of the button and invoke the Tcl command associated with the checkbutton, if there is one. The return value is the return value from the Tcl command, or an empty string if there is no command associated with the checkbutton. This command is ignored if the checkbutton's state is **disabled**.

**pathName select**

Selects the checkbutton and sets the associated variable to its ``on'' value.

**pathName toggle**

Toggles the selection state of the button, redisplaying it and modifying its associated variable to reflect the new state.

**BINDINGS**

Tk automatically creates class bindings for checkbuttons that give them the following default behavior:

1. On Unix systems, a checkbutton activates whenever the mouse passes over it and deactivates whenever the mouse leaves the checkbutton. On Mac and Windows systems, when mouse button 1 is pressed over a checkbutton, the button activates whenever the mouse pointer is inside the button, and deactivates whenever the mouse pointer leaves the button.

2. When mouse button 1 is pressed over a checkbutton, it is invoked (its selection state toggles and the command associated with the button is invoked, if there is one).

3. When a checkbutton has the input focus, the space key causes the checkbutton to be invoked. Under Windows, there are additional key bindings: plus (+) and equal (=) select the button, and minus (−) deselects the button.

If the checkbutton's state is **disabled** then none of the above actions occur: the checkbutton is completely non-responsive.

The behavior of checkbuttons can be changed by defining new bindings for individual widgets or by redefining the class bindings.
SEE ALSO

button, options, radiobutton

KEYWORDS

checkbutton, widget

Copyright © 1990−1994 The Regents of the University of California.
Copyright © 1994−1996 Sun Microsystems, Inc.
Copyright © 1995−1997 Roger E. Critchlow Jr.
NAME

clipboard – Manipulate Tk clipboard

SYNOPSIS

clipboard option ?arg arg ...?

DESCRIPTION

clipboard clear ?–displayof window?
clipboard append ?–displayof window? ?–format format? ?–type type? ?–? data
clipboard get ?–displayof window? ?–type type?

EXAMPLES

SEE ALSO

KEYWORDS
The format argument specifies the representation that should be used to transmit the selection to the requester (the second column of Table 2 of the ICCCM), and defaults to STRING. If format is STRING, the selection is transmitted as 8-bit ASCII characters. If format is ATOM, then the data is divided into fields separated by white space; each field is converted to its atom value, and the 32-bit atom value is transmitted instead of the atom name. For any other format, data is divided into fields separated by white space and each field is converted to a 32-bit integer; an array of integers is transmitted to the selection requester. Note that strings passed to clipboard append are concatenated before conversion, so the caller must take care to ensure appropriate spacing across string boundaries. All items appended to the clipboard with the same type must have the same format.

The format argument is needed only for compatibility with clipboard requesters that don’t use Tk. If the Tk toolkit is being used to retrieve the CLIPBOARD selection then the value is converted back to a string at the requesting end, so format is irrelevant.

A -- argument may be specified to mark the end of options; the next argument will always be used as data. This feature may be convenient if, for example, data starts with a -.

```
clipboard get ?-displayof window? ?-type type?
```

Retrieve data from the clipboard on window's display. window defaults to ".". Type specifies the form in which the data is to be returned and should be an atom name such as STRING or FILE_NAME. Type defaults to STRING. This command is equivalent to selection get -selection CLIPBOARD.

**EXAMPLES**

Get the current contents of the clipboard.

```
if {[catch {clipboard get} contents]} {
    # There were no clipboard contents at all
}
```

Set the clipboard to contain a fixed string.

```
clipboard clear
clipboard append "some fixed string"
```

**SEE ALSO**

selection

**KEYWORDS**

clear, format, clipboard, append, selection, type
colors

NAME

colors – symbolic color names recognized by Tk

DESCRIPTION

Tk recognizes many symbolic color names (eg, red) when specifying colors. The symbolic names recognized by Tk and their 8-bit RGB values are:

<table>
<thead>
<tr>
<th>Color</th>
<th>RGB Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice blue</td>
<td>240 248 255</td>
</tr>
<tr>
<td>AliceBlue</td>
<td>240 248 255</td>
</tr>
<tr>
<td>antique white</td>
<td>250 235 215</td>
</tr>
<tr>
<td>AntiqueWhite</td>
<td>250 235 215</td>
</tr>
<tr>
<td>AntiqueWhite1</td>
<td>255 239 219</td>
</tr>
<tr>
<td>AntiqueWhite2</td>
<td>238 223 204</td>
</tr>
<tr>
<td>AntiqueWhite3</td>
<td>205 192 176</td>
</tr>
<tr>
<td>AntiqueWhite4</td>
<td>139 131 120</td>
</tr>
<tr>
<td>aquamarine</td>
<td>127 255 212</td>
</tr>
<tr>
<td>aquamarine1</td>
<td>127 255 212</td>
</tr>
<tr>
<td>aquamarine2</td>
<td>118 238 198</td>
</tr>
<tr>
<td>aquamarine3</td>
<td>102 205 170</td>
</tr>
<tr>
<td>aquamarine4</td>
<td>69  139 116</td>
</tr>
<tr>
<td>azure</td>
<td>240 255 255</td>
</tr>
<tr>
<td>azure1</td>
<td>240 255 255</td>
</tr>
<tr>
<td>azure2</td>
<td>224 238 238</td>
</tr>
<tr>
<td>azure3</td>
<td>193 205 205</td>
</tr>
<tr>
<td>azure4</td>
<td>131 139 139</td>
</tr>
<tr>
<td>beige</td>
<td>245 245 220</td>
</tr>
<tr>
<td>bisque</td>
<td>255 228 196</td>
</tr>
<tr>
<td>bisque1</td>
<td>255 228 196</td>
</tr>
<tr>
<td>bisque2</td>
<td>238 213 183</td>
</tr>
<tr>
<td>bisque3</td>
<td>205 183 158</td>
</tr>
<tr>
<td>bisque4</td>
<td>139 125 107</td>
</tr>
<tr>
<td>black</td>
<td>0    0    0</td>
</tr>
<tr>
<td>blanched almond</td>
<td>255 235 205</td>
</tr>
<tr>
<td>BlanchedAlmond</td>
<td>255 235 205</td>
</tr>
<tr>
<td>blue</td>
<td>0    0 255</td>
</tr>
<tr>
<td>blue violet</td>
<td>138  43  226</td>
</tr>
<tr>
<td>blue1</td>
<td>0    0 255</td>
</tr>
<tr>
<td>blue2</td>
<td>0    0 238</td>
</tr>
<tr>
<td>blue3</td>
<td>0    0 205</td>
</tr>
<tr>
<td>blue4</td>
<td>0    0 139</td>
</tr>
<tr>
<td>BlueViolet</td>
<td>138  43  226</td>
</tr>
<tr>
<td>brown</td>
<td>165  42  42</td>
</tr>
<tr>
<td>brown1</td>
<td>255  64  64</td>
</tr>
<tr>
<td>brown2</td>
<td>238  59  59</td>
</tr>
<tr>
<td>brown3</td>
<td>205  51  51</td>
</tr>
<tr>
<td>brown4</td>
<td>139  35  35</td>
</tr>
<tr>
<td>burlywood</td>
<td>222 184 135</td>
</tr>
<tr>
<td>Color Name</td>
<td>Red</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----</td>
</tr>
<tr>
<td>burlywood1</td>
<td>255</td>
</tr>
<tr>
<td>burlywood2</td>
<td>238</td>
</tr>
<tr>
<td>burlywood3</td>
<td>205</td>
</tr>
<tr>
<td>burlywood4</td>
<td>139</td>
</tr>
<tr>
<td>cadet blue</td>
<td>95</td>
</tr>
<tr>
<td>CadetBlue</td>
<td>95</td>
</tr>
<tr>
<td>CadetBlue1</td>
<td>152</td>
</tr>
<tr>
<td>CadetBlue2</td>
<td>142</td>
</tr>
<tr>
<td>CadetBlue3</td>
<td>122</td>
</tr>
<tr>
<td>CadetBlue4</td>
<td>83</td>
</tr>
<tr>
<td>chartreuse</td>
<td>127</td>
</tr>
<tr>
<td>chartreuse1</td>
<td>127</td>
</tr>
<tr>
<td>chartreuse2</td>
<td>118</td>
</tr>
<tr>
<td>chartreuse3</td>
<td>102</td>
</tr>
<tr>
<td>chartreuse4</td>
<td>69</td>
</tr>
<tr>
<td>chocolate</td>
<td>210</td>
</tr>
<tr>
<td>chocolate1</td>
<td>255</td>
</tr>
<tr>
<td>chocolate2</td>
<td>238</td>
</tr>
<tr>
<td>chocolate3</td>
<td>205</td>
</tr>
<tr>
<td>chocolate4</td>
<td>139</td>
</tr>
<tr>
<td>coral</td>
<td>255</td>
</tr>
<tr>
<td>coral1</td>
<td>255</td>
</tr>
<tr>
<td>coral2</td>
<td>238</td>
</tr>
<tr>
<td>coral3</td>
<td>205</td>
</tr>
<tr>
<td>coral4</td>
<td>139</td>
</tr>
<tr>
<td>cornflower blue</td>
<td>100</td>
</tr>
<tr>
<td>CornflowerBlue</td>
<td>100</td>
</tr>
<tr>
<td>cornsilk</td>
<td>255</td>
</tr>
<tr>
<td>cornsilk1</td>
<td>255</td>
</tr>
<tr>
<td>cornsilk2</td>
<td>238</td>
</tr>
<tr>
<td>cornsilk3</td>
<td>205</td>
</tr>
<tr>
<td>cornsilk4</td>
<td>139</td>
</tr>
<tr>
<td>cyan</td>
<td>0</td>
</tr>
<tr>
<td>cyan1</td>
<td>0</td>
</tr>
<tr>
<td>cyan2</td>
<td>0</td>
</tr>
<tr>
<td>cyan3</td>
<td>0</td>
</tr>
<tr>
<td>cyan4</td>
<td>0</td>
</tr>
<tr>
<td>dark blue</td>
<td>0</td>
</tr>
<tr>
<td>dark cyan</td>
<td>0</td>
</tr>
<tr>
<td>dark goldenrod</td>
<td>184</td>
</tr>
<tr>
<td>dark gray</td>
<td>169</td>
</tr>
<tr>
<td>dark green</td>
<td>0</td>
</tr>
<tr>
<td>dark grey</td>
<td>169</td>
</tr>
<tr>
<td>dark khaki</td>
<td>189</td>
</tr>
<tr>
<td>dark magenta</td>
<td>139</td>
</tr>
<tr>
<td>dark olive green</td>
<td>85</td>
</tr>
<tr>
<td>dark orange</td>
<td>255</td>
</tr>
<tr>
<td>dark orchid</td>
<td>153</td>
</tr>
<tr>
<td>dark red</td>
<td>139</td>
</tr>
<tr>
<td>dark salmon</td>
<td>233</td>
</tr>
<tr>
<td>dark sea green</td>
<td>143</td>
</tr>
<tr>
<td>dark slate blue</td>
<td>72</td>
</tr>
<tr>
<td>dark slate gray</td>
<td>47</td>
</tr>
<tr>
<td>dark slate grey</td>
<td>47</td>
</tr>
<tr>
<td>dark turquoise</td>
<td>0</td>
</tr>
</tbody>
</table>
dark violet                148     0   211
DarkBlue                    0     0   139
DarkCyan                   0   139   139
DarkGoldenrod              184   134    11
DarkGoldenrod1              255   185   15
DarkGoldenrod2              238   173   14
DarkGoldenrod3              205   149   12
DarkGoldenrod4              139  101    8
DarkGray                    169   169   169
DarkGreen                   0   100    0
DarkGrey                   169   169   169
DarkKhaki                  189  183   107
DarkMagenta                139     0   139
DarkOliveGreen              85   107    47
DarkOliveGreen1            202   255   112
DarkOliveGreen2            188  238   104
DarkOliveGreen3            162   205    90
DarkOliveGreen4            110  139    61
DarkOrange                 255  140    0
DarkOrange1                 255  127    0
DarkOrange2                 238  118    0
DarkOrange3                 205  102    0
DarkOrange4                 139   69    0
DarkOrchid                 153   50  204
DarkOrchid1                 191  62  255
DarkOrchid2                 178  58  238
DarkOrchid3                 154   50  205
DarkOrchid4                 104   34  139
DarkRed                    139     0    0
DarkSalmon                  233  150   122
DarkSeaGreen                143  188   143
DarkSeaGreen1               193  255   193
DarkSeaGreen2               180  238   180
DarkSeaGreen3               155  205   155
DarkSeaGreen4               105  139   105
DarkSlateBlue                72   61   139
DarkSlateGray                47   79    79
DarkSlateGray1              151  255   255
DarkSlateGray2              141  238   238
DarkSlateGray3              121  205   205
DarkSlateGray4              82  139   139
DarkSlateGrey                47   79    79
DarkTurquoise               0   206   209
DarkViolet                 148     0   211
deep pink                   255   20   147
deep sky blue               0   191   255
DeepPink                    255   20   147
DeepPink1                   255   20   147
DeepPink2                   238   18   137
DeepPink3                   205   16   118
DeepPink4                   139   10    80
DeepSkyBlue                 0   191   255
DeepSkyBlue1                0   191   255
DeepSkyBlue2                0   178   238
DeepSkyBlue3                0   154   205
<table>
<thead>
<tr>
<th></th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepSkyBlue4</td>
<td>0</td>
<td>104</td>
<td>139</td>
</tr>
<tr>
<td>dim gray</td>
<td>105</td>
<td>105</td>
<td>105</td>
</tr>
<tr>
<td>dim grey</td>
<td>105</td>
<td>105</td>
<td>105</td>
</tr>
<tr>
<td>DimGray</td>
<td>105</td>
<td>105</td>
<td>105</td>
</tr>
<tr>
<td>DimGrey</td>
<td>105</td>
<td>105</td>
<td>105</td>
</tr>
<tr>
<td>dodger blue</td>
<td>30</td>
<td>144</td>
<td>255</td>
</tr>
<tr>
<td>DodgerBlue</td>
<td>30</td>
<td>144</td>
<td>255</td>
</tr>
<tr>
<td>DodgerBlue1</td>
<td>30</td>
<td>144</td>
<td>255</td>
</tr>
<tr>
<td>DodgerBlue2</td>
<td>28</td>
<td>134</td>
<td>238</td>
</tr>
<tr>
<td>DodgerBlue3</td>
<td>24</td>
<td>116</td>
<td>205</td>
</tr>
<tr>
<td>DodgerBlue4</td>
<td>16</td>
<td>78</td>
<td>139</td>
</tr>
<tr>
<td>firebrick</td>
<td>178</td>
<td>34</td>
<td>34</td>
</tr>
<tr>
<td>firebrick1</td>
<td>255</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>firebrick2</td>
<td>238</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>firebrick3</td>
<td>205</td>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>firebrick4</td>
<td>139</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>floral white</td>
<td>255</td>
<td>250</td>
<td>240</td>
</tr>
<tr>
<td>FloralWhite</td>
<td>255</td>
<td>250</td>
<td>240</td>
</tr>
<tr>
<td>forest green</td>
<td>34</td>
<td>139</td>
<td>34</td>
</tr>
<tr>
<td>ForestGreen</td>
<td>34</td>
<td>139</td>
<td>34</td>
</tr>
<tr>
<td>gainsboro</td>
<td>220</td>
<td>220</td>
<td>220</td>
</tr>
<tr>
<td>ghost white</td>
<td>248</td>
<td>248</td>
<td>255</td>
</tr>
<tr>
<td>GhostWhite</td>
<td>248</td>
<td>248</td>
<td>255</td>
</tr>
<tr>
<td>gold</td>
<td>255</td>
<td>215</td>
<td>0</td>
</tr>
<tr>
<td>gold1</td>
<td>255</td>
<td>215</td>
<td>0</td>
</tr>
<tr>
<td>gold2</td>
<td>238</td>
<td>201</td>
<td>0</td>
</tr>
<tr>
<td>gold3</td>
<td>205</td>
<td>173</td>
<td>0</td>
</tr>
<tr>
<td>gold4</td>
<td>139</td>
<td>117</td>
<td>0</td>
</tr>
<tr>
<td>goldenrod</td>
<td>218</td>
<td>165</td>
<td>32</td>
</tr>
<tr>
<td>goldenrod1</td>
<td>255</td>
<td>193</td>
<td>37</td>
</tr>
<tr>
<td>goldenrod2</td>
<td>238</td>
<td>180</td>
<td>34</td>
</tr>
<tr>
<td>goldenrod3</td>
<td>205</td>
<td>155</td>
<td>29</td>
</tr>
<tr>
<td>goldenrod4</td>
<td>139</td>
<td>105</td>
<td>20</td>
</tr>
<tr>
<td>gray</td>
<td>190</td>
<td>190</td>
<td>190</td>
</tr>
<tr>
<td>gray0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>gray1</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>gray2</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>gray3</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>gray4</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>gray5</td>
<td>13</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>gray6</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>gray7</td>
<td>18</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>gray8</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>gray9</td>
<td>23</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>gray10</td>
<td>26</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>gray11</td>
<td>28</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>gray12</td>
<td>31</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>gray13</td>
<td>33</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>gray14</td>
<td>36</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td>gray15</td>
<td>38</td>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>gray16</td>
<td>41</td>
<td>41</td>
<td>41</td>
</tr>
<tr>
<td>gray17</td>
<td>43</td>
<td>43</td>
<td>43</td>
</tr>
<tr>
<td>gray18</td>
<td>46</td>
<td>46</td>
<td>46</td>
</tr>
<tr>
<td>gray19</td>
<td>48</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Color</td>
<td>RGB Values</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray75</td>
<td>191 191 191</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray76</td>
<td>194 194 194</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray77</td>
<td>196 196 196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray78</td>
<td>199 199 199</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray79</td>
<td>201 201 201</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray80</td>
<td>204 204 204</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray81</td>
<td>207 207 207</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray82</td>
<td>209 209 209</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray83</td>
<td>212 212 212</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray84</td>
<td>214 214 214</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray85</td>
<td>217 217 217</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray86</td>
<td>219 219 219</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray87</td>
<td>222 222 222</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray88</td>
<td>224 224 224</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray89</td>
<td>227 227 227</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray90</td>
<td>229 229 229</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray91</td>
<td>232 232 232</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray92</td>
<td>235 235 235</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray93</td>
<td>237 237 237</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray94</td>
<td>240 240 240</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray95</td>
<td>242 242 242</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray96</td>
<td>245 245 245</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray97</td>
<td>247 247 247</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray98</td>
<td>250 250 250</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray99</td>
<td>252 252 252</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gray100</td>
<td>255 255 255</td>
<td></td>
<td></td>
</tr>
<tr>
<td>green</td>
<td>0 255 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>green yellow</td>
<td>173 255 47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>green1</td>
<td>0 255 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>green2</td>
<td>0 238 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>green3</td>
<td>0 205 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>green4</td>
<td>0 139 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GreenYellow</td>
<td>173 255 47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey</td>
<td>190 190 190</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey0</td>
<td>0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey1</td>
<td>3 3 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey2</td>
<td>5 5 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey3</td>
<td>8 8 8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey4</td>
<td>10 10 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey5</td>
<td>13 13 13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey6</td>
<td>15 15 15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey7</td>
<td>18 18 18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey8</td>
<td>20 20 20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey9</td>
<td>23 23 23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey10</td>
<td>26 26 26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey11</td>
<td>28 28 28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey12</td>
<td>31 31 31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey13</td>
<td>33 33 33</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey14</td>
<td>36 36 36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey15</td>
<td>38 38 38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey16</td>
<td>41 41 41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey17</td>
<td>43 43 43</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey18</td>
<td>46 46 46</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey19</td>
<td>48 48 48</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey20</td>
<td>51 51 51</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grey21</td>
<td>54</td>
<td>54</td>
<td>54</td>
</tr>
<tr>
<td>--------</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>grey22</td>
<td>56</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>grey23</td>
<td>59</td>
<td>59</td>
<td>59</td>
</tr>
<tr>
<td>grey24</td>
<td>61</td>
<td>61</td>
<td>61</td>
</tr>
<tr>
<td>grey25</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>grey26</td>
<td>66</td>
<td>66</td>
<td>66</td>
</tr>
<tr>
<td>grey27</td>
<td>69</td>
<td>69</td>
<td>69</td>
</tr>
<tr>
<td>grey28</td>
<td>71</td>
<td>71</td>
<td>71</td>
</tr>
<tr>
<td>grey29</td>
<td>74</td>
<td>74</td>
<td>74</td>
</tr>
<tr>
<td>grey30</td>
<td>77</td>
<td>77</td>
<td>77</td>
</tr>
<tr>
<td>grey31</td>
<td>79</td>
<td>79</td>
<td>79</td>
</tr>
<tr>
<td>grey32</td>
<td>82</td>
<td>82</td>
<td>82</td>
</tr>
<tr>
<td>grey33</td>
<td>84</td>
<td>84</td>
<td>84</td>
</tr>
<tr>
<td>grey34</td>
<td>87</td>
<td>87</td>
<td>87</td>
</tr>
<tr>
<td>grey35</td>
<td>89</td>
<td>89</td>
<td>89</td>
</tr>
<tr>
<td>grey36</td>
<td>92</td>
<td>92</td>
<td>92</td>
</tr>
<tr>
<td>grey37</td>
<td>94</td>
<td>94</td>
<td>94</td>
</tr>
<tr>
<td>grey38</td>
<td>97</td>
<td>97</td>
<td>97</td>
</tr>
<tr>
<td>grey39</td>
<td>99</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>grey40</td>
<td>102</td>
<td>102</td>
<td>102</td>
</tr>
<tr>
<td>grey41</td>
<td>105</td>
<td>105</td>
<td>105</td>
</tr>
<tr>
<td>grey42</td>
<td>107</td>
<td>107</td>
<td>107</td>
</tr>
<tr>
<td>grey43</td>
<td>110</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>grey44</td>
<td>112</td>
<td>112</td>
<td>112</td>
</tr>
<tr>
<td>grey45</td>
<td>115</td>
<td>115</td>
<td>115</td>
</tr>
<tr>
<td>grey46</td>
<td>117</td>
<td>117</td>
<td>117</td>
</tr>
<tr>
<td>grey47</td>
<td>120</td>
<td>120</td>
<td>120</td>
</tr>
<tr>
<td>grey48</td>
<td>122</td>
<td>122</td>
<td>122</td>
</tr>
<tr>
<td>grey49</td>
<td>125</td>
<td>125</td>
<td>125</td>
</tr>
<tr>
<td>grey50</td>
<td>127</td>
<td>127</td>
<td>127</td>
</tr>
<tr>
<td>grey51</td>
<td>130</td>
<td>130</td>
<td>130</td>
</tr>
<tr>
<td>grey52</td>
<td>133</td>
<td>133</td>
<td>133</td>
</tr>
<tr>
<td>grey53</td>
<td>135</td>
<td>135</td>
<td>135</td>
</tr>
<tr>
<td>grey54</td>
<td>138</td>
<td>138</td>
<td>138</td>
</tr>
<tr>
<td>grey55</td>
<td>140</td>
<td>140</td>
<td>140</td>
</tr>
<tr>
<td>grey56</td>
<td>143</td>
<td>143</td>
<td>143</td>
</tr>
<tr>
<td>grey57</td>
<td>145</td>
<td>145</td>
<td>145</td>
</tr>
<tr>
<td>grey58</td>
<td>148</td>
<td>148</td>
<td>148</td>
</tr>
<tr>
<td>grey59</td>
<td>150</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>grey60</td>
<td>153</td>
<td>153</td>
<td>153</td>
</tr>
<tr>
<td>grey61</td>
<td>156</td>
<td>156</td>
<td>156</td>
</tr>
<tr>
<td>grey62</td>
<td>158</td>
<td>158</td>
<td>158</td>
</tr>
<tr>
<td>grey63</td>
<td>161</td>
<td>161</td>
<td>161</td>
</tr>
<tr>
<td>grey64</td>
<td>163</td>
<td>163</td>
<td>163</td>
</tr>
<tr>
<td>grey65</td>
<td>166</td>
<td>166</td>
<td>166</td>
</tr>
<tr>
<td>grey66</td>
<td>168</td>
<td>168</td>
<td>168</td>
</tr>
<tr>
<td>grey67</td>
<td>171</td>
<td>171</td>
<td>171</td>
</tr>
<tr>
<td>grey68</td>
<td>173</td>
<td>173</td>
<td>173</td>
</tr>
<tr>
<td>grey69</td>
<td>176</td>
<td>176</td>
<td>176</td>
</tr>
<tr>
<td>grey70</td>
<td>179</td>
<td>179</td>
<td>179</td>
</tr>
<tr>
<td>grey71</td>
<td>181</td>
<td>181</td>
<td>181</td>
</tr>
<tr>
<td>grey72</td>
<td>184</td>
<td>184</td>
<td>184</td>
</tr>
<tr>
<td>grey73</td>
<td>186</td>
<td>186</td>
<td>186</td>
</tr>
<tr>
<td>grey74</td>
<td>189</td>
<td>189</td>
<td>189</td>
</tr>
<tr>
<td>grey75</td>
<td>191</td>
<td>191</td>
<td>191</td>
</tr>
</tbody>
</table>

colors 429
<table>
<thead>
<tr>
<th>Color</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>grey76</td>
<td>194</td>
<td>194</td>
<td>194</td>
</tr>
<tr>
<td>grey77</td>
<td>196</td>
<td>196</td>
<td>196</td>
</tr>
<tr>
<td>grey78</td>
<td>199</td>
<td>199</td>
<td>199</td>
</tr>
<tr>
<td>grey79</td>
<td>201</td>
<td>201</td>
<td>201</td>
</tr>
<tr>
<td>grey80</td>
<td>204</td>
<td>204</td>
<td>204</td>
</tr>
<tr>
<td>grey81</td>
<td>207</td>
<td>207</td>
<td>207</td>
</tr>
<tr>
<td>grey82</td>
<td>209</td>
<td>209</td>
<td>209</td>
</tr>
<tr>
<td>grey83</td>
<td>212</td>
<td>212</td>
<td>212</td>
</tr>
<tr>
<td>grey84</td>
<td>214</td>
<td>214</td>
<td>214</td>
</tr>
<tr>
<td>grey85</td>
<td>217</td>
<td>217</td>
<td>217</td>
</tr>
<tr>
<td>grey86</td>
<td>219</td>
<td>219</td>
<td>219</td>
</tr>
<tr>
<td>grey87</td>
<td>222</td>
<td>222</td>
<td>222</td>
</tr>
<tr>
<td>grey88</td>
<td>224</td>
<td>224</td>
<td>224</td>
</tr>
<tr>
<td>grey89</td>
<td>227</td>
<td>227</td>
<td>227</td>
</tr>
<tr>
<td>grey90</td>
<td>229</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>grey91</td>
<td>232</td>
<td>232</td>
<td>232</td>
</tr>
<tr>
<td>grey92</td>
<td>235</td>
<td>235</td>
<td>235</td>
</tr>
<tr>
<td>grey93</td>
<td>237</td>
<td>237</td>
<td>237</td>
</tr>
<tr>
<td>grey94</td>
<td>240</td>
<td>240</td>
<td>240</td>
</tr>
<tr>
<td>grey95</td>
<td>242</td>
<td>242</td>
<td>242</td>
</tr>
<tr>
<td>grey96</td>
<td>245</td>
<td>245</td>
<td>245</td>
</tr>
<tr>
<td>grey97</td>
<td>247</td>
<td>247</td>
<td>247</td>
</tr>
<tr>
<td>grey98</td>
<td>250</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>grey99</td>
<td>252</td>
<td>252</td>
<td>252</td>
</tr>
<tr>
<td>grey100</td>
<td>255</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>honeydew</td>
<td>240</td>
<td>255</td>
<td>240</td>
</tr>
<tr>
<td>honeydew1</td>
<td>240</td>
<td>255</td>
<td>240</td>
</tr>
<tr>
<td>honeydew2</td>
<td>224</td>
<td>238</td>
<td>224</td>
</tr>
<tr>
<td>honeydew3</td>
<td>193</td>
<td>205</td>
<td>193</td>
</tr>
<tr>
<td>honeydew4</td>
<td>131</td>
<td>139</td>
<td>131</td>
</tr>
<tr>
<td>hot pink</td>
<td>255</td>
<td>105</td>
<td>180</td>
</tr>
<tr>
<td>HotPink</td>
<td>255</td>
<td>105</td>
<td>180</td>
</tr>
<tr>
<td>HotPink1</td>
<td>255</td>
<td>110</td>
<td>180</td>
</tr>
<tr>
<td>HotPink2</td>
<td>238</td>
<td>106</td>
<td>167</td>
</tr>
<tr>
<td>HotPink3</td>
<td>205</td>
<td>96</td>
<td>144</td>
</tr>
<tr>
<td>HotPink4</td>
<td>139</td>
<td>58</td>
<td>98</td>
</tr>
<tr>
<td>indian red</td>
<td>205</td>
<td>92</td>
<td>92</td>
</tr>
<tr>
<td>IndianRed</td>
<td>205</td>
<td>92</td>
<td>92</td>
</tr>
<tr>
<td>IndianRed1</td>
<td>255</td>
<td>106</td>
<td>106</td>
</tr>
<tr>
<td>IndianRed2</td>
<td>238</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>IndianRed3</td>
<td>205</td>
<td>85</td>
<td>85</td>
</tr>
<tr>
<td>IndianRed4</td>
<td>139</td>
<td>58</td>
<td>58</td>
</tr>
<tr>
<td>ivory</td>
<td>255</td>
<td>255</td>
<td>240</td>
</tr>
<tr>
<td>ivory1</td>
<td>255</td>
<td>255</td>
<td>240</td>
</tr>
<tr>
<td>ivory2</td>
<td>238</td>
<td>238</td>
<td>224</td>
</tr>
<tr>
<td>ivory3</td>
<td>205</td>
<td>205</td>
<td>193</td>
</tr>
<tr>
<td>ivory4</td>
<td>139</td>
<td>139</td>
<td>131</td>
</tr>
<tr>
<td>khaki</td>
<td>240</td>
<td>230</td>
<td>140</td>
</tr>
<tr>
<td>khaki1</td>
<td>255</td>
<td>246</td>
<td>143</td>
</tr>
<tr>
<td>khaki2</td>
<td>238</td>
<td>230</td>
<td>133</td>
</tr>
<tr>
<td>khaki3</td>
<td>205</td>
<td>198</td>
<td>115</td>
</tr>
<tr>
<td>khaki4</td>
<td>139</td>
<td>134</td>
<td>78</td>
</tr>
<tr>
<td>lavender</td>
<td>230</td>
<td>230</td>
<td>250</td>
</tr>
<tr>
<td>lavender blush</td>
<td>255</td>
<td>240</td>
<td>245</td>
</tr>
<tr>
<td>Color</td>
<td>RGB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lavender Blush</td>
<td>255 240 245</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lavender Blush 1</td>
<td>255 240 245</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lavender Blush 2</td>
<td>238 224 229</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lavender Blush 3</td>
<td>205 193 197</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lavender Blush 4</td>
<td>139 131 134</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lawn green</td>
<td>124 252 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lawn Green</td>
<td>124 252 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lemon chiffon</td>
<td>255 250 205</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lemon Chiffon</td>
<td>255 250 205</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lemon Chiffon 1</td>
<td>255 250 205</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lemon Chiffon 2</td>
<td>238 233 191</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lemon Chiffon 3</td>
<td>205 201 165</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lemon Chiffon 4</td>
<td>139 137 112</td>
<td></td>
<td></td>
</tr>
<tr>
<td>light blue</td>
<td>173 216 230</td>
<td></td>
<td></td>
</tr>
<tr>
<td>light coral</td>
<td>240 128 128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>light cyan</td>
<td>224 255 255</td>
<td></td>
<td></td>
</tr>
<tr>
<td>light goldenrod</td>
<td>238 221 130</td>
<td></td>
<td></td>
</tr>
<tr>
<td>light goldenrod yellow</td>
<td>250 250 210</td>
<td></td>
<td></td>
</tr>
<tr>
<td>light gray</td>
<td>211 211 211</td>
<td></td>
<td></td>
</tr>
<tr>
<td>light green</td>
<td>144 238 144</td>
<td></td>
<td></td>
</tr>
<tr>
<td>light grey</td>
<td>211 211 211</td>
<td></td>
<td></td>
</tr>
<tr>
<td>light pink</td>
<td>255 182 193</td>
<td></td>
<td></td>
</tr>
<tr>
<td>light salmon</td>
<td>255 160 122</td>
<td></td>
<td></td>
</tr>
<tr>
<td>light sea green</td>
<td>32 178 170</td>
<td></td>
<td></td>
</tr>
<tr>
<td>light sky blue</td>
<td>135 206 250</td>
<td></td>
<td></td>
</tr>
<tr>
<td>light slate blue</td>
<td>132 112 255</td>
<td></td>
<td></td>
</tr>
<tr>
<td>light slate gray</td>
<td>119 136 153</td>
<td></td>
<td></td>
</tr>
<tr>
<td>light slate grey</td>
<td>119 136 153</td>
<td></td>
<td></td>
</tr>
<tr>
<td>light steel blue</td>
<td>176 196 222</td>
<td></td>
<td></td>
</tr>
<tr>
<td>light yellow</td>
<td>255 255 224</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Blue</td>
<td>173 216 230</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Blue 1</td>
<td>191 239 255</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Blue 2</td>
<td>178 223 238</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Blue 3</td>
<td>154 192 205</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Blue 4</td>
<td>104 131 139</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Coral</td>
<td>240 128 128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Cyan</td>
<td>224 255 255</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Cyan 1</td>
<td>224 255 255</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Cyan 2</td>
<td>209 238 238</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Cyan 3</td>
<td>180 205 205</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Cyan 4</td>
<td>122 139 139</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Goldenrod</td>
<td>238 221 130</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Goldenrod 1</td>
<td>255 236 139</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Goldenrod 2</td>
<td>238 220 130</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Goldenrod 3</td>
<td>205 190 112</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Goldenrod 4</td>
<td>139 129 76</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Goldenrod Yellow</td>
<td>250 250 210</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Gray</td>
<td>211 211 211</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Green</td>
<td>144 238 144</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Grey</td>
<td>211 211 211</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Pink</td>
<td>255 182 193</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Pink 1</td>
<td>255 174 185</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Pink 2</td>
<td>238 162 173</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Pink 3</td>
<td>205 140 149</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Pink 4</td>
<td>139 95 101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Color Name</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>LightSalmon</td>
<td>255</td>
<td>160</td>
<td>122</td>
</tr>
<tr>
<td>LightSalmon1</td>
<td>255</td>
<td>160</td>
<td>122</td>
</tr>
<tr>
<td>LightSalmon2</td>
<td>238</td>
<td>149</td>
<td>114</td>
</tr>
<tr>
<td>LightSalmon3</td>
<td>205</td>
<td>129</td>
<td>98</td>
</tr>
<tr>
<td>LightSalmon4</td>
<td>139</td>
<td>87</td>
<td>66</td>
</tr>
<tr>
<td>LightSeaGreen</td>
<td>32</td>
<td>178</td>
<td>170</td>
</tr>
<tr>
<td>LightSkyBlue</td>
<td>135</td>
<td>206</td>
<td>250</td>
</tr>
<tr>
<td>LightSkyBlue1</td>
<td>176</td>
<td>226</td>
<td>255</td>
</tr>
<tr>
<td>LightSkyBlue2</td>
<td>164</td>
<td>211</td>
<td>238</td>
</tr>
<tr>
<td>LightSkyBlue3</td>
<td>141</td>
<td>182</td>
<td>205</td>
</tr>
<tr>
<td>LightSkyBlue4</td>
<td>96</td>
<td>123</td>
<td>139</td>
</tr>
<tr>
<td>LightSlateBlue</td>
<td>132</td>
<td>112</td>
<td>255</td>
</tr>
<tr>
<td>LightSlateGray</td>
<td>119</td>
<td>136</td>
<td>153</td>
</tr>
<tr>
<td>LightSlateGrey</td>
<td>119</td>
<td>136</td>
<td>153</td>
</tr>
<tr>
<td>LightSteelBlue</td>
<td>176</td>
<td>196</td>
<td>222</td>
</tr>
<tr>
<td>LightSteelBlue1</td>
<td>202</td>
<td>225</td>
<td>255</td>
</tr>
<tr>
<td>LightSteelBlue2</td>
<td>188</td>
<td>210</td>
<td>238</td>
</tr>
<tr>
<td>LightSteelBlue3</td>
<td>162</td>
<td>181</td>
<td>205</td>
</tr>
<tr>
<td>LightSteelBlue4</td>
<td>110</td>
<td>123</td>
<td>139</td>
</tr>
<tr>
<td>LightYellow</td>
<td>255</td>
<td>255</td>
<td>224</td>
</tr>
<tr>
<td>LightYellow1</td>
<td>255</td>
<td>255</td>
<td>224</td>
</tr>
<tr>
<td>LightYellow2</td>
<td>238</td>
<td>238</td>
<td>209</td>
</tr>
<tr>
<td>LightYellow3</td>
<td>205</td>
<td>205</td>
<td>180</td>
</tr>
<tr>
<td>LightYellow4</td>
<td>139</td>
<td>139</td>
<td>122</td>
</tr>
<tr>
<td>lime green</td>
<td>50</td>
<td>205</td>
<td>50</td>
</tr>
<tr>
<td>LimeGreen</td>
<td>50</td>
<td>205</td>
<td>50</td>
</tr>
<tr>
<td>linen</td>
<td>250</td>
<td>240</td>
<td>230</td>
</tr>
<tr>
<td>magenta</td>
<td>255</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>magenta1</td>
<td>255</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>magenta2</td>
<td>238</td>
<td>0</td>
<td>238</td>
</tr>
<tr>
<td>magenta3</td>
<td>205</td>
<td>0</td>
<td>205</td>
</tr>
<tr>
<td>magenta4</td>
<td>139</td>
<td>0</td>
<td>139</td>
</tr>
<tr>
<td>maroon</td>
<td>176</td>
<td>48</td>
<td>96</td>
</tr>
<tr>
<td>maroon1</td>
<td>255</td>
<td>52</td>
<td>179</td>
</tr>
<tr>
<td>maroon2</td>
<td>238</td>
<td>48</td>
<td>167</td>
</tr>
<tr>
<td>maroon3</td>
<td>205</td>
<td>41</td>
<td>144</td>
</tr>
<tr>
<td>maroon4</td>
<td>139</td>
<td>28</td>
<td>98</td>
</tr>
<tr>
<td>medium aquamarine</td>
<td>102</td>
<td>205</td>
<td>170</td>
</tr>
<tr>
<td>medium blue</td>
<td>0</td>
<td>0</td>
<td>205</td>
</tr>
<tr>
<td>medium orchid</td>
<td>186</td>
<td>85</td>
<td>211</td>
</tr>
<tr>
<td>medium purple</td>
<td>147</td>
<td>112</td>
<td>219</td>
</tr>
<tr>
<td>medium sea green</td>
<td>60</td>
<td>179</td>
<td>113</td>
</tr>
<tr>
<td>medium slate blue</td>
<td>123</td>
<td>104</td>
<td>238</td>
</tr>
<tr>
<td>medium spring green</td>
<td>0</td>
<td>250</td>
<td>154</td>
</tr>
<tr>
<td>medium turquoise</td>
<td>72</td>
<td>209</td>
<td>204</td>
</tr>
<tr>
<td>medium violet red</td>
<td>199</td>
<td>21</td>
<td>133</td>
</tr>
<tr>
<td>MediumAquamarine</td>
<td>102</td>
<td>205</td>
<td>170</td>
</tr>
<tr>
<td>MediumBlue</td>
<td>0</td>
<td>0</td>
<td>205</td>
</tr>
<tr>
<td>MediumOrchid</td>
<td>186</td>
<td>85</td>
<td>211</td>
</tr>
<tr>
<td>MediumOrchid1</td>
<td>224</td>
<td>102</td>
<td>255</td>
</tr>
<tr>
<td>MediumOrchid2</td>
<td>209</td>
<td>95</td>
<td>238</td>
</tr>
<tr>
<td>MediumOrchid3</td>
<td>180</td>
<td>82</td>
<td>205</td>
</tr>
<tr>
<td>MediumOrchid4</td>
<td>122</td>
<td>55</td>
<td>139</td>
</tr>
<tr>
<td>MediumPurple</td>
<td>147</td>
<td>112</td>
<td>219</td>
</tr>
<tr>
<td>MediumPurple1</td>
<td>171</td>
<td>130</td>
<td>255</td>
</tr>
<tr>
<td>Color Name</td>
<td>RGB Values</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MediumPurple2</td>
<td>159 121 238</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MediumPurple3</td>
<td>137 104 205</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MediumPurple4</td>
<td>93  71  139</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MediumSeaGreen</td>
<td>60 179 113</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MediumSlateBlue</td>
<td>123 104 238</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MediumSpringGreen</td>
<td>0   250 154</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MediumTurquoise</td>
<td>72 209 204</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MediumVioletRed</td>
<td>199 21 133</td>
<td></td>
<td></td>
</tr>
<tr>
<td>midnight blue</td>
<td>25 25 112</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MidnightBlue</td>
<td>25 25 112</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mint cream</td>
<td>245 255 250</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MintCream</td>
<td>245 255 250</td>
<td></td>
<td></td>
</tr>
<tr>
<td>misty rose</td>
<td>255 228 225</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MistyRose</td>
<td>255 228 225</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MistyRose1</td>
<td>255 228 225</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MistyRose2</td>
<td>238 213 210</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MistyRose3</td>
<td>205 183 181</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MistyRose4</td>
<td>139 125 123</td>
<td></td>
<td></td>
</tr>
<tr>
<td>moccasin</td>
<td>255 228 181</td>
<td></td>
<td></td>
</tr>
<tr>
<td>navajo white</td>
<td>255 222 173</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NavajoWhite</td>
<td>255 222 173</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NavajoWhite1</td>
<td>255 222 173</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NavajoWhite2</td>
<td>238 207 161</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NavajoWhite3</td>
<td>205 179 139</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NavajoWhite4</td>
<td>139 121 94</td>
<td></td>
<td></td>
</tr>
<tr>
<td>navy</td>
<td>0   0  128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>navy blue</td>
<td>0   0  128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NavyBlue</td>
<td>0   0  128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>old lace</td>
<td>253 245 230</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OldLace</td>
<td>253 245 230</td>
<td></td>
<td></td>
</tr>
<tr>
<td>olive drab</td>
<td>107 142  35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OliveDrab</td>
<td>107 142  35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OliveDrab1</td>
<td>192 255  62</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OliveDrab2</td>
<td>179 238  58</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OliveDrab3</td>
<td>154 205  50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OliveDrab4</td>
<td>105 139  34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>orange</td>
<td>255 165  0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>orange red</td>
<td>255  69  0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>orange1</td>
<td>255 165  0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>orange2</td>
<td>238 154  0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>orange3</td>
<td>205 133  0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>orange4</td>
<td>139  90  0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OrangeRed</td>
<td>255  69  0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OrangeRed1</td>
<td>255  69  0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OrangeRed2</td>
<td>238  64  0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OrangeRed3</td>
<td>205  55  0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OrangeRed4</td>
<td>139   37  0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>orchid</td>
<td>218  112 214</td>
<td></td>
<td></td>
</tr>
<tr>
<td>orchid1</td>
<td>255  131 250</td>
<td></td>
<td></td>
</tr>
<tr>
<td>orchid2</td>
<td>238  122 233</td>
<td></td>
<td></td>
</tr>
<tr>
<td>orchid3</td>
<td>205  105 201</td>
<td></td>
<td></td>
</tr>
<tr>
<td>orchid4</td>
<td>139   71  137</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pale goldenrod</td>
<td>238 232 170</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pale green</td>
<td>152 251 152</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pale turquoise</td>
<td>175 238 238</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**colors**
<table>
<thead>
<tr>
<th>Name</th>
<th>RGB Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>pale violet red</td>
<td>219 112 147</td>
</tr>
<tr>
<td>PaleGoldenrod</td>
<td>238 232 170</td>
</tr>
<tr>
<td>PaleGreen</td>
<td>152 251 152</td>
</tr>
<tr>
<td>PaleGreen1</td>
<td>154 255 154</td>
</tr>
<tr>
<td>PaleGreen2</td>
<td>144 238 144</td>
</tr>
<tr>
<td>PaleGreen3</td>
<td>124 205 124</td>
</tr>
<tr>
<td>PaleGreen4</td>
<td>84 139 84</td>
</tr>
<tr>
<td>PaleTurquoise</td>
<td>175 238 238</td>
</tr>
<tr>
<td>PaleTurquoise1</td>
<td>187 255 255</td>
</tr>
<tr>
<td>PaleTurquoise2</td>
<td>174 238 238</td>
</tr>
<tr>
<td>PaleTurquoise3</td>
<td>150 205 205</td>
</tr>
<tr>
<td>PaleTurquoise4</td>
<td>102 139 139</td>
</tr>
<tr>
<td>PaleVioletRed</td>
<td>219 112 147</td>
</tr>
<tr>
<td>PaleVioletRed1</td>
<td>255 130 171</td>
</tr>
<tr>
<td>PaleVioletRed2</td>
<td>238 121 159</td>
</tr>
<tr>
<td>PaleVioletRed3</td>
<td>205 104 127</td>
</tr>
<tr>
<td>PaleVioletRed4</td>
<td>139 71 93</td>
</tr>
<tr>
<td>papaya whip</td>
<td>255 239 213</td>
</tr>
<tr>
<td>PapayaWhip</td>
<td>255 239 213</td>
</tr>
<tr>
<td>peach puff</td>
<td>255 218 185</td>
</tr>
<tr>
<td>PeachPuff</td>
<td>255 218 185</td>
</tr>
<tr>
<td>PeachPuff1</td>
<td>255 218 185</td>
</tr>
<tr>
<td>PeachPuff2</td>
<td>238 203 173</td>
</tr>
<tr>
<td>PeachPuff3</td>
<td>205 175 149</td>
</tr>
<tr>
<td>PeachPuff4</td>
<td>139 119 101</td>
</tr>
<tr>
<td>peru</td>
<td>205 133 63</td>
</tr>
<tr>
<td>pink</td>
<td>255 192 203</td>
</tr>
<tr>
<td>pink1</td>
<td>255 181 197</td>
</tr>
<tr>
<td>pink2</td>
<td>238 169 184</td>
</tr>
<tr>
<td>pink3</td>
<td>205 145 158</td>
</tr>
<tr>
<td>pink4</td>
<td>139 99 108</td>
</tr>
<tr>
<td>plum</td>
<td>221 160 221</td>
</tr>
<tr>
<td>plum1</td>
<td>255 187 255</td>
</tr>
<tr>
<td>plum2</td>
<td>238 174 238</td>
</tr>
<tr>
<td>plum3</td>
<td>205 150 205</td>
</tr>
<tr>
<td>plum4</td>
<td>139 102 139</td>
</tr>
<tr>
<td>powder blue</td>
<td>176 224 230</td>
</tr>
<tr>
<td>PowderBlue</td>
<td>176 224 230</td>
</tr>
<tr>
<td>purple</td>
<td>160 32 240</td>
</tr>
<tr>
<td>purple1</td>
<td>155 48 255</td>
</tr>
<tr>
<td>purple2</td>
<td>145 44 238</td>
</tr>
<tr>
<td>purple3</td>
<td>125 38 205</td>
</tr>
<tr>
<td>purple4</td>
<td>85 26 139</td>
</tr>
<tr>
<td>red</td>
<td>255 0 0</td>
</tr>
<tr>
<td>red1</td>
<td>255 0 0</td>
</tr>
<tr>
<td>red2</td>
<td>238 0 0</td>
</tr>
<tr>
<td>red3</td>
<td>205 0 0</td>
</tr>
<tr>
<td>red4</td>
<td>139 0 0</td>
</tr>
<tr>
<td>rosy brown</td>
<td>188 143 143</td>
</tr>
<tr>
<td>RosyBrown</td>
<td>188 143 143</td>
</tr>
<tr>
<td>RosyBrown1</td>
<td>255 193 193</td>
</tr>
<tr>
<td>RosyBrown2</td>
<td>238 180 180</td>
</tr>
<tr>
<td>RosyBrown3</td>
<td>205 155 155</td>
</tr>
<tr>
<td>RosyBrown4</td>
<td>139 105 105</td>
</tr>
</tbody>
</table>

Colors: 434
<table>
<thead>
<tr>
<th>Color Name</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>royal blue</td>
<td>65</td>
<td>105</td>
<td>225</td>
</tr>
<tr>
<td>RoyalBlue</td>
<td>65</td>
<td>105</td>
<td>225</td>
</tr>
<tr>
<td>RoyalBlue1</td>
<td>72</td>
<td>118</td>
<td>255</td>
</tr>
<tr>
<td>RoyalBlue2</td>
<td>67</td>
<td>110</td>
<td>238</td>
</tr>
<tr>
<td>RoyalBlue3</td>
<td>58</td>
<td>95</td>
<td>205</td>
</tr>
<tr>
<td>RoyalBlue4</td>
<td>39</td>
<td>64</td>
<td>139</td>
</tr>
<tr>
<td>saddle brown</td>
<td>139</td>
<td>69</td>
<td>19</td>
</tr>
<tr>
<td>SaddleBrown</td>
<td>139</td>
<td>69</td>
<td>19</td>
</tr>
<tr>
<td>salmon</td>
<td>250</td>
<td>128</td>
<td>114</td>
</tr>
<tr>
<td>salmon1</td>
<td>255</td>
<td>140</td>
<td>105</td>
</tr>
<tr>
<td>salmon2</td>
<td>238</td>
<td>130</td>
<td>98</td>
</tr>
<tr>
<td>salmon3</td>
<td>205</td>
<td>112</td>
<td>84</td>
</tr>
<tr>
<td>salmon4</td>
<td>139</td>
<td>76</td>
<td>57</td>
</tr>
<tr>
<td>sandy brown</td>
<td>244</td>
<td>164</td>
<td>96</td>
</tr>
<tr>
<td>SandyBrown</td>
<td>244</td>
<td>164</td>
<td>96</td>
</tr>
<tr>
<td>sea green</td>
<td>46</td>
<td>139</td>
<td>87</td>
</tr>
<tr>
<td>SeaGreen</td>
<td>46</td>
<td>139</td>
<td>87</td>
</tr>
<tr>
<td>SeaGreen1</td>
<td>84</td>
<td>255</td>
<td>159</td>
</tr>
<tr>
<td>SeaGreen2</td>
<td>78</td>
<td>238</td>
<td>148</td>
</tr>
<tr>
<td>SeaGreen3</td>
<td>67</td>
<td>205</td>
<td>128</td>
</tr>
<tr>
<td>SeaGreen4</td>
<td>46</td>
<td>139</td>
<td>87</td>
</tr>
<tr>
<td>seashell</td>
<td>255</td>
<td>245</td>
<td>238</td>
</tr>
<tr>
<td>seashell1</td>
<td>255</td>
<td>245</td>
<td>238</td>
</tr>
<tr>
<td>seashell2</td>
<td>238</td>
<td>229</td>
<td>222</td>
</tr>
<tr>
<td>seashell3</td>
<td>205</td>
<td>197</td>
<td>191</td>
</tr>
<tr>
<td>seashell4</td>
<td>139</td>
<td>134</td>
<td>130</td>
</tr>
<tr>
<td>sienna</td>
<td>160</td>
<td>82</td>
<td>45</td>
</tr>
<tr>
<td>sienna1</td>
<td>255</td>
<td>130</td>
<td>71</td>
</tr>
<tr>
<td>sienna2</td>
<td>238</td>
<td>121</td>
<td>66</td>
</tr>
<tr>
<td>sienna3</td>
<td>205</td>
<td>104</td>
<td>57</td>
</tr>
<tr>
<td>sienna4</td>
<td>139</td>
<td>71</td>
<td>38</td>
</tr>
<tr>
<td>sky blue</td>
<td>135</td>
<td>206</td>
<td>235</td>
</tr>
<tr>
<td>SkyBlue</td>
<td>135</td>
<td>206</td>
<td>235</td>
</tr>
<tr>
<td>SkyBlue1</td>
<td>135</td>
<td>206</td>
<td>255</td>
</tr>
<tr>
<td>SkyBlue2</td>
<td>126</td>
<td>192</td>
<td>238</td>
</tr>
<tr>
<td>SkyBlue3</td>
<td>108</td>
<td>166</td>
<td>205</td>
</tr>
<tr>
<td>SkyBlue4</td>
<td>74</td>
<td>112</td>
<td>139</td>
</tr>
<tr>
<td>slate blue</td>
<td>106</td>
<td>90</td>
<td>205</td>
</tr>
<tr>
<td>slate gray</td>
<td>112</td>
<td>128</td>
<td>144</td>
</tr>
<tr>
<td>slate grey</td>
<td>112</td>
<td>128</td>
<td>144</td>
</tr>
<tr>
<td>SlateBlue</td>
<td>106</td>
<td>90</td>
<td>205</td>
</tr>
<tr>
<td>SlateBlue1</td>
<td>131</td>
<td>111</td>
<td>255</td>
</tr>
<tr>
<td>SlateBlue2</td>
<td>122</td>
<td>103</td>
<td>238</td>
</tr>
<tr>
<td>SlateBlue3</td>
<td>105</td>
<td>89</td>
<td>205</td>
</tr>
<tr>
<td>SlateBlue4</td>
<td>71</td>
<td>60</td>
<td>139</td>
</tr>
<tr>
<td>SlateGray</td>
<td>112</td>
<td>128</td>
<td>144</td>
</tr>
<tr>
<td>SlateGray1</td>
<td>198</td>
<td>226</td>
<td>255</td>
</tr>
<tr>
<td>SlateGray2</td>
<td>185</td>
<td>211</td>
<td>238</td>
</tr>
<tr>
<td>SlateGray3</td>
<td>159</td>
<td>182</td>
<td>205</td>
</tr>
<tr>
<td>SlateGray4</td>
<td>108</td>
<td>123</td>
<td>139</td>
</tr>
<tr>
<td>SlateGrey</td>
<td>112</td>
<td>128</td>
<td>144</td>
</tr>
<tr>
<td>snow</td>
<td>255</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>snow1</td>
<td>255</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>snow2</td>
<td>238</td>
<td>233</td>
<td>233</td>
</tr>
<tr>
<td>snow3</td>
<td>205</td>
<td>201</td>
<td>201</td>
</tr>
</tbody>
</table>
snow4                      139   137   137
spring green                 0   255   127
SpringGreen                  0   255   127
SpringGreen1                 0   255   127
SpringGreen2                 0   238   118
SpringGreen3                 0   205   102
SpringGreen4                 0   139    69
steel blue                  70   130   180
SteelBlue                   70   130   180
SteelBlue1                  99   184   255
SteelBlue2                  92   172   238
SteelBlue3                  79   148   205
SteelBlue4                  54   100   139
tan                        210   180   140
tan1                       255   165    79
tan2                       238   154    73
tan3                       205   133    63
tan4                        139    90    43
thistle                    216   191   216
thistle1                   255   225   255
thistle2                   238   210   238
thistle3                   205   181   205
thistle4                   139   123   139
tomato                     255   99    71
tomato1                    255   99    71
tomato2                    238   92    66
tomato3                    205   79    57
tomato4                    139   54    38
turquoise                   64   224   208
turquoise1                  0   245   255
turquoise2                  0   229   238
turquoise3                  0   197   205
turquoise4                  0   134   139
violet                     238   130   238
violet red                 208    32   144
VioletRed                  208    32   144
VioletRed1                 255    62   150
VioletRed2                 238    58   140
VioletRed3                 205    50   120
VioletRed4                 139    34    82
wheat                      245   222   179
wheat1                     255   231   186
wheat2                     238   216   174
wheat3                     205   186   150
wheat4                     139   126   102
white                       255   255   255
white smoke                245   245   245
WhiteSmoke                 245   245   245
yellow                     255   255    0
yellow green                154   205    50
yellow1                    255   255    0
yellow2                    238   238    0
yellow3                    205   205    0
yellow4                    139   139    0
YellowGreen                 154   205    50
KEYWORDS

color, option

Copyright © 2003 ActiveState Corporation.
Copyright © 1995−1997 Roger E. Critchlow Jr.
console

NAME

console – Control the console on systems without a real console

SYNOPSIS

console subcommand ?arg ...? 

DESCRIPTION

console eval script
console hide
console show
console title ?string?

ACCESS TO THE MAIN INTERPRETER

consoleinterp eval script
consoleinterp record script

ADDITIONAL TRAP CALLS

DEFAULT BINDINGS

EXAMPLE

KEYWORDS

SEE ALSO

NAME

console – Control the console on systems without a real console

SYNOPSIS

console subcommand ?arg ...? 

DESCRIPTION

The console window is a replacement for a real console to allow input and output on the standard I/O channels on platforms that do not have a real console. It is implemented as a separate interpreter with the Tk toolkit loaded, and control over this interpreter is given through the console command. The behaviour of the console window is defined mainly through the contents of the console.tcl file in the Tk library (or the Console resource on Macintosh systems.)

console eval script
Evaluate the script argument as a Tcl script in the console interpreter. The normal interpreter is accessed through the consoleinterp command in the console interpreter.

console hide
Hide the console window from view. Precisely equivalent to withdrawing the . window in the console interpreter.
**console show**
Display the console window. Precisely equivalent to deiconifying the . window in the console interpreter.

**console title ?string?**
Query or modify the title of the console window. If string is not specified, queries the title of the console window, and sets the title of the console window to string otherwise. Precisely equivalent to using the **wm title** command in the console interpreter.

---

**ACCESS TO THE MAIN INTERPRETER**

The **consoleinterp** command in the console interpreter allows scripts to be evaluated in the main interpreter. It supports two subcommands: **eval** and **record**.

- **consoleinterp eval script**
  Evaluates script as a Tcl script at the global level in the main interpreter.

- **consoleinterp record script**
  Records and evaluates script as a Tcl script at the global level in the main interpreter as if script had been typed in at the console.

---

**ADDITIONAL TRAP CALLS**

There are several additional commands in the console interpreter that are called in response to activity in the main interpreter. These are documented here for completeness only; they form part of the internal implementation of the console and are likely to change or be modified without warning.

Output to the console from the main interpreter via the stdout and stderr channels is handled by invoking the **tk::ConsoleOutput** command in the console interpreter with two arguments. The first argument is the name of the channel being written to, and the second argument is the string being written to the channel (after encoding and end-of-line translation processing has been performed.)

When the . window of the main interpreter is destroyed, the **tk::ConsoleExit** command in the console interpreter is called (assuming the console interpreter has not already been deleted itself, that is.)

---

**DEFAULT BINDINGS**

The default script creates a console window (implemented using a text widget) that has the following behaviour:

1. Pressing the tab key inserts a TAB character (as defined by the Tcl \t escape.)
2. Pressing the return key causes the current line (if complete by the rules of **info complete**) to be passed to the main interpreter for evaluation.
3. 

---
Pressing the delete key deletes the selected text (if any text is selected) or the character to the right of the cursor (if not at the end of the line.)

Pressing the backspace key deletes the selected text (if any text is selected) or the character to the left of the cursor (of not at the start of the line.)

Pressing either Control+A or the home key causes the cursor to go to the start of the line (but after the prompt, if a prompt is present on the line.)

Pressing either Control+E or the end key causes the cursor to go to the end of the line.

Pressing either Control+P or the up key causes the previous entry in the command history to be selected.

Pressing either Control+N or the down key causes the next entry in the command history to be selected.

Pressing either Control+B or the left key causes the cursor to move one character backward as long as the cursor is not at the prompt.

Pressing either Control+F or the right key causes the cursor to move one character forward.

Pressing F9 rebuilds the console window by destroying all its children and reloading the Tcl script that defined the console's behaviour.

Most other behaviour is the same as a conventional text widget except for the way that the <<Cut>> event is handled identically to the <<Copy>> event.

EXAMPLE

Not all platforms have the console command, so debugging code often has the following code fragment in it so output produced by puts can be seen while during development:

\[
\text{catch \{console show\}}
\]

KEYWORDS

class, interpreter, window, interactive, output channels

SEE ALSO

destroy, fconfigure, history, interp, puts, text, wm

Copyright ' 2001 Donal K. Fellows
Copyright ' 1995-1997 Roger E. Critchlow Jr.
cursors

NAME

cursors − mouse cursors available in Tk

DESCRIPTION

The -cursor widget option allows a Tk programmer to change the mouse cursor for a particular widget. The cursor names recognized by Tk on all platforms are:

X_cursor
arrow
based_arrow_down
based_arrow_up
boat
bogosity
bottom_left_corner
bottom_right_corner
bottom_side
bottom_tee
box_spiral
center_ptr
circle
clock
coffee_mug
cross
cross_reverse
crosshair
diamond_cross
dot
dotbox
double_arrow
draft_large
draft_small
draped_box
exchange
fleur
gobbler
gumby
hand1
hand2
heart
icon
iron_cross
left_ptr
left_side
left_tee
leftbutton
ll_angle
lr_angle
PORTABILITY ISSUES

Windows

On Windows systems, the following cursors are mapped to native cursors:

---

arrow
center_ptr
crosshair
fleur
ibeam
icon
sb_h_double_arrow
sb_v_double_arrow
watch
xterm
And the following additional cursors are available:

no
starting
size
size_ne_sw
size_ns
size_nw_se
size_we
uparrow
wait

The no cursor can be specified to eliminate the cursor.

Macintosh
On Macintosh systems, the following cursors are mapped to native cursors:

arrow
cross
crosshair
ibeam
plus
watch
xterm

And the following additional cursors are available:

text
cross-hair

KEYWORDS

cursor, option

Copyright ' 1995-1997 Roger E. Critchlow Jr.
destroy

NAME

destroy – Destroy one or more windows

SYNOPSIS

destroy ?window window ...?

DESCRIPTION

This command deletes the windows given by the window arguments, plus all of their descendants. If a window ``.'' is deleted then the entire application will be destroyed. The windows are destroyed in order, and if an error occurs in destroying a window the command aborts without destroying the remaining windows. No error is returned if window does not exist.

EXAMPLE

Destroy all checkbuttons that are direct children of the given widget:

```tcl
proc killCheckbuttonChildren {parent} {
    foreach w [winfo children $parent] {
        if {[winfo class $w] eq "Checkbutton"} {
            destroy $w
        }
    }
}
```

KEYWORDS

application, destroy, window

Copyright ' 1990 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

entry – Create and manipulate entry widgets

SYNOPSIS

entry pathName ?options?

STANDARD OPTIONS

-background or -bg, background, Background
-borderwidth or -bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-exportselection, exportSelection, ExportSelection
-font, font, Font
-foreground or -fg, foreground, Foreground
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-insertbackground, insertBackground, Foreground
-insertborderwidth, insertBorderWidth, BorderWidth
-insertofftime, insertOffTime, OffTime
-inserton, insertOnTime, OnTime
-insertwidth, insertWidth, InsertWidth
-justify, justify, Justify
-relief, relief, Relief
-selectbackground, selectBackground, Foreground
-selectborderwidth, selectBorderWidth, BorderWidth
-selectforeground, selectForeground, Background
-takefocus, takeFocus, TakeFocus
-textvariable, textVariable, Variable
-xscrollcommand, xScrollCommand, ScrollCommand

WIDGET-SPECIFIC OPTIONS

-disabledbackground, disabledBackground, DisabledBackground
-disabledforeground, disabledForeground, DisabledForeground
-invalidcommand or -invcmd, invalidCommand, InvalidCommand
-readonlybackground, readonlyBackground, ReadonlyBackground
-show, show, Show
-state, state, State
-validate, validate, Validate
-validatecommand or -vcmd, validateCommand, ValidateCommand
-width, width, Width

DESCRIPTION

VALIDATION

none

focus
focusin
focusout
key
all
%d
%i
%P
%s
%S
%v
%V
%W

WIDGET COMMAND

number
anchor
ded
insert
sel.first
sel.last
@number
pathName bbox index
pathName cget option
pathName configure ?option? ?value option value ...?
pathName delete first ?last?
pathName get
pathName icursor index
pathName index index
pathName insert index string
pathName scan option args
  pathName scan mark x
  pathName scan dragto x
pathName selection option arg
  pathName selection adjust index
  pathName selection clear
  pathName selection from index
  pathName selection present
  pathName selection range start end
  pathName selection to index
pathName validate
pathName xview args
  pathName xview
  pathName xview index
  pathName xview moveto fraction
  pathName xview scroll number what

DEFAULT BINDINGS
KEYWORDS

NAME

entry – Create and manipulate entry widgets

SYNOPSIS

entry pathName ?options?

STANDARD OPTIONS

−background or −bg, background, Background
−borderwidth or −bd, borderWidth, BorderWidth
−cursor, cursor, Cursor
−exportselection, exportSelection, ExportSelection
−font, font, Font
−foreground or −fg, foreground, Foreground
−highlightbackground, highlightBackground, HighlightBackground
−highlightcolor, highlightColor, HighlightColor
−highlightthickness, highlightThickness, HighlightThickness
−insertbackground, insertBackground, Foreground
−insertborderwidth, insertBorderWidth, BorderWidth
−insertofftime, insertOffTime, OffTime
−insertontime, insertOnTime, OnTime
−insertwidth, insertWidth, InsertWidth
−justify, justify, Justify
−relief, relief, Relief
−selectbackground, selectBackground, Foreground
−selectborderwidth, selectBorderWidth, BorderWidth
−selectforeground, selectForeground, Background
−takefocus, takeFocus, TakeFocus
−textvariable, textVariable, Variable
−xscrollcommand, xScrollCommand, ScrollCommand

WIDGET–SPECIFIC OPTIONS

Command–Line Name: −disabledbackground
Database Name: disabledBackground
Database Class: DisabledBackground

  Specifies the background color to use when the entry is disabled. If this option is the empty string, the
  normal background color is used.

Command–Line Name: −disabledforeground
Database Name: disabledForeground
**Database Class: DisabledForeground**

Specifies the foreground color to use when the entry is disabled. If this option is the empty string, the normal foreground color is used.

**Command-Line Name:** −invalidcommand or −invcmd

**Database Name:** invalidCommand

**Database Class:** InvalidCommand

Specifies a script to eval when validateCommand returns 0. Setting it to {} disables this feature (the default). The best use of this option is to set it to bell. See Validation below for more information.

**Command-Line Name:** −readonlybackground

**Database Name:** readonlyBackground

**Database Class:** ReadonlyBackground

Specifies the background color to use when the entry is readonly. If this option is the empty string, the normal background color is used.

**Command-Line Name:** −show

**Database Name:** show

**Database Class:** Show

If this option is specified, then the true contents of the entry are not displayed in the window. Instead, each character in the entry’s value will be displayed as the first character in the value of this option, such as “*”. This is useful, for example, if the entry is to be used to enter a password. If characters in the entry are selected and copied elsewhere, the information copied will be what is displayed, not the true contents of the entry.

**Command-Line Name:** −state

**Database Name:** state

**Database Class:** State

Specifies one of three states for the entry: normal, disabled, or readonly. If the entry is readonly, then the value may not be changed using widget commands and no insertion cursor will be displayed, even if the input focus is in the widget; the contents of the widget may still be selected. If the entry is disabled, the value may not be changed, no insertion cursor will be displayed, the contents will not be selectable, and the entry may be displayed in a different color, depending on the values of the −disabledforeground and −disabledbackground options.

**Command-Line Name:** −validate

**Database Name:** validate

**Database Class:** Validate

Specifies the mode in which validation should operate: none, focus, focusin, focusout, key, or all. It defaults to none. When you want validation, you must explicitly state which mode you wish to use. See Validation below for more.

**Command-Line Name:** −validatecommand or −vcmd

**Database Name:** validateCommand

**Database Class:** ValidateCommand

Specifies a script to eval when you want to validate the input into the entry widget. Setting it to {} disables this feature (the default). This command must return a valid Tcl boolean value. If it returns 0 (or the valid Tcl boolean equivalent) then it means you reject the new edition and it will not occur and the invalidCommand will be evaluated if it is set. If it returns 1, then the new edition occurs. See Validation below for more information.

**Command-Line Name:** −width
Database Name: width
Database Class: Width

Specifies an integer value indicating the desired width of the entry window, in average-size characters of the widget's font. If the value is less than or equal to zero, the widget picks a size just large enough to hold its current text.

DESCRIPTION

The entry command creates a new window (given by the pathName argument) and makes it into an entry widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the entry such as its colors, font, and relief. The entry command returns its pathName argument. At the time this command is invoked, there must not exist a window named pathName, but pathName's parent must exist.

An entry is a widget that displays a one-line text string and allows that string to be edited using widget commands described below, which are typically bound to keystrokes and mouse actions. When first created, an entry's string is empty. A portion of the entry may be selected as described below. If an entry is exporting its selection (see the exportSelection option), then it will observe the standard X11 protocols for handling the selection; entry selections are available as type STRING. Entries also observe the standard Tk rules for dealing with the input focus. When an entry has the input focus it displays an insertion cursor to indicate where new characters will be inserted.

Entries are capable of displaying strings that are too long to fit entirely within the widget's window. In this case, only a portion of the string will be displayed; commands described below may be used to change the view in the window. Entries use the standard xScrollCommand mechanism for interacting with scrollbars (see the description of the xScrollCommand option for details). They also support scanning, as described below.

VALIDATION

Validation works by setting the validateCommand option to a script which will be evaluated according to the validate option as follows:

none
Default. This means no validation will occur.

focus
validateCommand will be called when the entry receives or loses focus.

focusin
validateCommand will be called when the entry receives focus.

focusout
validateCommand will be called when the entry loses focus.

key
validateCommand will be called when the entry is edited.

all
validateCommand will be called for all above conditions.
It is possible to perform percent substitutions on the `validateCommand` and `invalidCommand`, just as you would in a `bind` script. The following substitutions are recognized:

%-d  Type of action: 1 for `insert`, 0 for `delete`, or −1 for focus, forced or textvariable validation.
%-i  Index of char string to be inserted/deleted, if any, otherwise −1.
%-P  The value of the entry if the edit is allowed. If you are configuring the entry widget to have a new textvariable, this will be the value of that textvariable.
%-s  The current value of entry prior to editing.
%-S  The text string being inserted/deleted, if any, {} otherwise.
%-v  The type of validation currently set.
%-V  The type of validation that triggered the callback (key, focusin, focusout, forced).
%-W  The name of the entry widget.

In general, the `textVariable` and `validateCommand` can be dangerous to mix. Any problems have been overcome so that using the `validateCommand` will not interfere with the traditional behavior of the entry widget. Using the `textVariable` for read-only purposes will never cause problems. The danger comes when you try set the `textVariable` to something that the `validateCommand` would not accept, which causes `validate` to become `none` (the `invalidCommand` will not be triggered). The same happens when an error occurs evaluating the `validateCommand`.

Primarily, an error will occur when the `validateCommand` or `invalidCommand` encounters an error in its script while evaluating or `validateCommand` does not return a valid Tcl boolean value. The `validate` option will also set itself to `none` when you edit the entry widget from within either the `validateCommand` or the `invalidCommand`. Such editions will override the one that was being validated. If you wish to edit the entry widget (for example set it to {}) during validation and still have the `validate` option set, you should include the command

```
after idle {%W config −validate %v}
```

in the `validateCommand` or `invalidCommand` (whichever one you were editing the entry widget from). It is also recommended to not set an associated `textVariable` during validation, as that can cause the entry widget to become out of sync with the `textVariable`.

**WIDGET COMMAND**

The `entry` command creates a new Tcl command whose name is `pathName`. This command may be used to invoke various operations on the widget. It has the following general form:
pathName option ?arg arg ...?

Option and the args determine the exact behavior of the command.

Many of the widget commands for entries take one or more indices as arguments. An index specifies a particular character in the entry's string, in any of the following ways:

number
   Specifies the character as a numerical index, where 0 corresponds to the first character in the string.

anchor
   Indicates the anchor point for the selection, which is set with the select from and select adjust widget commands.

del
   Indicates the character just after the last one in the entry's string. This is equivalent to specifying a numerical index equal to the length of the entry's string.

insert
   Indicates the character adjacent to and immediately following the insertion cursor.

sel.first
   Indicates the first character in the selection. It is an error to use this form if the selection isn't in the entry window.

sel.last
   Indicates the character just after the last one in the selection. It is an error to use this form if the selection isn't in the entry window.

@number
   In this form, number is treated as an x−coordinate in the entry's window; the character spanning that x−coordinate is used. For example, ```@0'' indicates the left−most character in the window.

Abbreviations may be used for any of the forms above, e.g. ``e'' or ``sel.f''. In general, out−of−range indices are automatically rounded to the nearest legal value.

The following commands are possible for entry widgets:

pathName bbox index
   Returns a list of four numbers describing the bounding box of the character given by index. The first two elements of the list give the x and y coordinates of the upper−left corner of the screen area covered by the character (in pixels relative to the widget) and the last two elements give the width and height of the character, in pixels. The bounding box may refer to a region outside the visible area of the window.

pathName cget option
   Returns the current value of the configuration option given by option. Option may have any of the values accepted by the entry command.

pathName configure ?option? ?value option value ...?
   Query or modify the configuration options of the widget. If no option is specified, returns a list describing all of the available options for pathName (see Tk_ConfigureInfo for information on the format of this list). If option is specified with no value, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no
option is specified). If one or more option−value pairs are specified, then the command modifies the
given widget option(s) to have the given value(s); in this case the command returns an empty string.
Option may have any of the values accepted by the entry command.

pathName delete first ?last?  
Delete one or more elements of the entry. First is the index of the first character to delete, and last is
the index of the character just after the last one to delete. If last isn't specified it defaults to first+1, i.e.
a single character is deleted. This command returns an empty string.

pathName get  
Returns the entry's string.

pathName icursor index  
Arrange for the insertion cursor to be displayed just before the character given by index. Returns an
empty string.

pathName index index  
Returns the numerical index corresponding to index.

pathName insert index string  
Insert the characters of string just before the character indicated by index. Returns an empty string.

pathName scan option args  
This command is used to implement scanning on entries. It has two forms, depending on option:

pathName scan mark x  
Records x and the current view in the entry window; used in conjunction with later scan
dragto commands. Typically this command is associated with a mouse button press in the
widget. It returns an empty string.

pathName scan dragto x  
This command computes the difference between its x argument and the x argument to the last
scan mark command for the widget. It then adjusts the view left or right by 10 times the
difference in x−coordinates. This command is typically associated with mouse motion events
in the widget, to produce the effect of dragging the entry at high speed through the window.
The return value is an empty string.

pathName selection option arg  
This command is used to adjust the selection within an entry. It has several forms, depending on
option:

pathName selection adjust index  
Locate the end of the selection nearest to the character given by index, and adjust that end of
the selection to be at index (i.e. including but not going beyond index). The other end of the
selection is made the anchor point for future select to commands. If the selection isn't
currently in the entry, then a new selection is created to include the characters between index
and the most recent selection anchor point, inclusive. Returns an empty string.

pathName selection clear  
Clear the selection if it is currently in this widget. If the selection isn't in this widget then the
command has no effect. Returns an empty string.

pathName selection from index  
Set the selection anchor point to just before the character given by index. Doesn't change the
selection. Returns an empty string.
pathName selection present
   Returns 1 if there is an are characters selected in the entry, 0 if nothing is selected.

pathName selection range start end
   Sets the selection to include the characters starting with the one indexed by start and ending
   with the one just before end. If end refers to the same character as start or an earlier one, then
   the entry's selection is cleared.

pathName selection to index
   If index is before the anchor point, set the selection to the characters from index up to but not
   including the anchor point. If index is the same as the anchor point, do nothing. If index is
   after the anchor point, set the selection to the characters from the anchor point up to but not
   including index. The anchor point is determined by the most recent select from or select
   adjust command in this widget. If the selection isn't in this widget then a new selection is
   created using the most recent anchor point specified for the widget. Returns an empty string.

pathName validate
   This command is used to force an evaluation of the validateCommand independent of the conditions
   specified by the validate option. This is done by temporarily setting the validate option to all. It
   returns 0 or 1.

pathName xview args
   This command is used to query and change the horizontal position of the text in the widget's window.
   It can take any of the following forms:

pathName xview
   Returns a list containing two elements. Each element is a real fraction between 0 and 1;
   together they describe the horizontal span that is visible in the window. For example, if the
   first element is .2 and the second element is .6, 20% of the entry's text is off-screen to the
   left, the middle 40% is visible in the window, and 40% of the text is off-screen to the right.
   These are the same values passed to scrollbars via the −xscrollcommand option.

pathName xview index
   Adjusts the view in the window so that the character given by index is displayed at the left
   edge of the window.

pathName xview moveto fraction
   Adjusts the view in the window so that the character fraction of the way through the text
   appears at the left edge of the window. Fraction must be a fraction between 0 and 1.

pathName xview scroll number what
   This command shifts the view in the window left or right according to number and what.
   Number must be an integer. What must be either units or pages or an abbreviation of one of
   these. If what is units, the view adjusts left or right by number average-width characters on
   the display; if it is pages then the view adjusts by number screenfuls. If number is negative
   then characters farther to the left become visible; if it is positive then characters farther to the
   right become visible.

DEFAULT BINDINGS

Tk automatically creates class bindings for entries that give them the following default behavior. In the
descriptions below, ``word'' refers to a contiguous group of letters, digits, or `_` characters, or any single
character other than these.

[1] Clicking mouse button 1 positions the insertion cursor just before the character underneath the mouse cursor, sets the input focus to this widget, and clears any selection in the widget. Dragging with mouse button 1 strokes out a selection between the insertion cursor and the character under the mouse.

[2] Double-clicking with mouse button 1 selects the word under the mouse and positions the insertion cursor at the end of the word. Dragging after a double click will stroke out a selection consisting of whole words.

[3] Triple-clicking with mouse button 1 selects all of the text in the entry and positions the insertion cursor at the end of the line.

[4] The ends of the selection can be adjusted by dragging with mouse button 1 while the Shift key is down; this will adjust the end of the selection that was nearest to the mouse cursor when button 1 was pressed. If the button is double-clicked before dragging then the selection will be adjusted in units of whole words.

[5] Clicking mouse button 1 with the Control key down will position the insertion cursor in the entry without affecting the selection.

[6] If any normal printing characters are typed in an entry, they are inserted at the point of the insertion cursor.

[7] The view in the entry can be adjusted by dragging with mouse button 2. If mouse button 2 is clicked without moving the mouse, the selection is copied into the entry at the position of the mouse cursor.

[8] If the mouse is dragged out of the entry on the left or right sides while button 1 is pressed, the entry will automatically scroll to make more text visible (if there is more text off-screen on the side where the mouse left the window).

[9] The Left and Right keys move the insertion cursor one character to the left or right; they also clear any selection in the entry and set the selection anchor. If Left or Right is typed with the Shift key down, then the insertion cursor moves and the selection is extended to include the new character. Control−Left and Control−Right move the insertion cursor by words, and Control−Shift−Left and Control−Shift−Right move the insertion cursor by words and also extend the selection. Control−b and Control−f behave the same as Left and Right, respectively. Meta−b and Meta−f behave the same as Control−Left and Control−Right, respectively.

[10] The Home key, or Control−a, will move the insertion cursor to the beginning of the entry and clear any selection in the entry. Shift−Home moves the insertion cursor to the beginning of the entry and also extends the selection to that point.

[11]
The End key, or Control–e, will move the insertion cursor to the end of the entry and clear any selection in the entry. Shift–End moves the cursor to the end and extends the selection to that point.

The Select key and Control–Space set the selection anchor to the position of the insertion cursor. They don't affect the current selection. Shift–Select and Control–Shift–Space adjust the selection to the current position of the insertion cursor, selecting from the anchor to the insertion cursor if there was not any selection previously.

Control–/ selects all the text in the entry.

Control–\ clears any selection in the entry.

The F16 key (labelled Copy on many Sun workstations) or Meta–w copies the selection in the widget to the clipboard, if there is a selection.

The F20 key (labelled Cut on many Sun workstations) or Control–w copies the selection in the widget to the clipboard and deletes the selection. If there is no selection in the widget then these keys have no effect.

The F18 key (labelled Paste on many Sun workstations) or Control–y inserts the contents of the clipboard at the position of the insertion cursor.

The Delete key deletes the selection, if there is one in the entry. If there is no selection, it deletes the character to the right of the insertion cursor.

The BackSpace key and Control–h delete the selection, if there is one in the entry. If there is no selection, it deletes the character to the left of the insertion cursor.

Control–d deletes the character to the right of the insertion cursor.

Meta–d deletes the word to the right of the insertion cursor.

Control–k deletes all the characters to the right of the insertion cursor.

Control–t reverses the order of the two characters to the right of the insertion cursor.

If the entry is disabled using the −state option, then the entry's view can still be adjusted and text in the entry can still be selected, but no insertion cursor will be displayed and no text modifications will take place except if the entry is linked to a variable using the −textvariable option, in which case any changes to the variable are reflected by the entry whatever the value of its −state option.

The behavior of entries can be changed by defining new bindings for individual widgets or by redefining the class bindings.
KEYWORDS

text, widget

Copyright © 1990−1994 The Regents of the University of California.
Copyright © 1994−1996 Sun Microsystems, Inc.
Copyright © 1995−1997 Roger E. Critchlow Jr.
event

NAME

event – Miscellaneous event facilities: define virtual events and generate events

SYNOPSIS

    event option ?arg arg ...?

DESCRIPTION

    event add <<virtual>> sequence ?sequence ...?
    event delete <<virtual>> ?sequence sequence ...?
    event generate window event ?option value option value ...?
    event info ?<<virtual>>?

EVENT FIELDS

    -above window
    -borderwidth size
    -button number
    -count number
    -delta number
    -detail detail
    -focus boolean
    -height size
    -keydown number
    -kevsm name
    -mode notify
    -override boolean
    -place where
    -root window
    -rootx coord
    -rooty coord
    -sendevent boolean
    -serial number
    -state state
    -subwindow window
    -time integer
    -warp boolean
    -width size
    -when when
        now
        tail
        head
        mark
    -x coord
    -y coord

VIRTUAL EVENT EXAMPLES
NAME

event – Miscellaneous event facilities: define virtual events and generate events

SYNOPSIS

event option ?arg arg ...?

DESCRIPTION

The event command provides several facilities for dealing with window system events, such as defining virtual events and synthesizing events. The command has several different forms, determined by the first argument. The following forms are currently supported:

event add <<virtual>> sequence ?sequence ...?

Associates the virtual event virtual with the physical event sequence(s) given by the sequence arguments, so that the virtual event will trigger whenever any one of the sequences occurs. Virtual may be any string value and sequence may have any of the values allowed for the sequence argument to the bind command. If virtual is already defined, the new physical event sequences add to the existing sequences for the event.

event delete <<virtual>> ?sequence sequence ...?

Deletes each of the sequences from those associated with the virtual event given by virtual. Virtual may be any string value and sequence may have any of the values allowed for the sequence argument to the bind command. Any sequences not currently associated with virtual are ignored. If no sequence argument is provided, all physical event sequences are removed for virtual, so that the virtual event will not trigger anymore.

event generate window event ?option value option value ...?

Generates a window event and arranges for it to be processed just as if it had come from the window system. Window gives the path name of the window for which the event will be generated; it may also be an identifier (such as returned by winfo id) as long as it is for a window in the current application. Event provides a basic description of the event, such as <Shift–Button–2> or <<Paste>>. If Window is empty the whole screen is meant, and coordinates are relative to the screen. Event may have any of the forms allowed for the sequence argument of the bind command except that it must consist of a single event pattern, not a sequence. Option–value pairs may be used to specify additional attributes of the event, such as the x and y mouse position; see EVENT FIELDS below. If the –when option is not specified, the event is processed immediately: all of the handlers for the event will complete before the event generate command returns. If the –when option is specified then it determines when the event is processed. Certain events, such as key events, require that the window has focus to receive the event properly.

event info ?<<virtual>>?

Returns information about virtual events. If the <<virtual>> argument is omitted, the return value is a list of all the virtual events that are currently defined. If <<virtual>> is specified then the return value
is a list whose elements are the physical event sequences currently defined for the given virtual event; if the virtual event is not defined then an empty string is returned.

**EVENT FIELDS**

The following options are supported for the `event generate` command. These correspond to the ```%``` expansions allowed in binding scripts for the `bind` command.

---

**above window**

*Window* specifies the *above* field for the event, either as a window path name or as an integer window id. Valid for *Configure* events. Corresponds to the `%a` substitution for binding scripts.

---

**borderwidth size**

*Size* must be a screen distance; it specifies the *border_width* field for the event. Valid for *Configure* events. Corresponds to the `%B` substitution for binding scripts.

---

**button number**

*Number* must be an integer; it specifies the *detail* field for a *ButtonPress* or *ButtonRelease* event, overriding any button number provided in the base *event* argument. Corresponds to the `%b` substitution for binding scripts.

---

**count number**

*Number* must be an integer; it specifies the *count* field for the event. Valid for *Expose* events. Corresponds to the `%c` substitution for binding scripts.

---

**delta number**

*Number* must be an integer; it specifies the *delta* field for the *MouseWheel* event. The *delta* refers to the direction and magnitude the mouse wheel was rotated. Note the value is not a screen distance but are units of motion in the mouse wheel. Typically these values are multiples of 120. For example, 120 should scroll the text widget up 4 lines and −240 would scroll the text widget down 8 lines. Of course, other widgets may define different behaviors for mouse wheel motion. This field corresponds to the `%D` substitution for binding scripts.

---

**detail detail**

*Detail* specifies the *detail* field for the event and must be one of the following:

- `NotifyAncestor`
- `NotifyNonlinearVirtual`
- `NotifyDetailNone`
- `NotifyPointer`
- `NotifyInferior`
- `NotifyPointerRoot`
- `NotifyNonlinear`
- `NotifyVirtual`

Valid for *Enter*, *Leave*, *FocusIn* and *FocusOut* events. Corresponds to the `%d` substitution for binding scripts.

---

**focus boolean**

*Boolean* must be a boolean value; it specifies the *focus* field for the event. Valid for *Enter* and *Leave* events. Corresponds to the `%f` substitution for binding scripts.

---

**height size**

*Size* must be a screen distance; it specifies the *height* field for the event. Valid for *Configure* events. Corresponds to the `%h` substitution for binding scripts.

---

**keycode number**
Number must be an integer; it specifies the keycode field for the event. Valid for KeyPress and KeyRelease events. Corresponds to the %k substitution for binding scripts.

---

**--keysym name**

Name must be the name of a valid keysym, such as g, space, or Return; its corresponding keycode value is used as the keycode field for event, overriding any detail specified in the base event argument. Valid for KeyPress and KeyRelease events. Corresponds to the %K substitution for binding scripts.

---

**--mode notify**

Notify specifies the mode field for the event and must be one of NotifyNormal, NotifyGrab, NotifyUngrab, or NotifyWhileGrabbed. Valid for Enter, Leave, FocusIn, and FocusOut events. Corresponds to the %m substitution for binding scripts.

---

**--override boolean**

Boolean must be a boolean value; it specifies the override_redirect field for the event. Valid for Map, Reparent, and Configure events. Corresponds to the %o substitution for binding scripts.

---

**--place where**

Where specifies the place field for the event; it must be either PlaceOnTop or PlaceOnBottom. Valid for Circulate events. Corresponds to the %p substitution for binding scripts.

---

**--root window**

Window must be either a window path name or an integer window identifier; it specifies the root field for the event. Valid for KeyPress, KeyRelease, ButtonPress, ButtonRelease, Enter, Leave, and Motion events. Corresponds to the %R substitution for binding scripts.

---

**--rootx coord**

Coord must be a screen distance; it specifies the x_root field for the event. Valid for KeyPress, KeyRelease, ButtonPress, ButtonRelease, Enter, Leave, and Motion events. Corresponds to the %X substitution for binding scripts.

---

**--rooty coord**

Coord must be a screen distance; it specifies the y_root field for the event. Valid for KeyPress, KeyRelease, ButtonPress, ButtonRelease, Enter, Leave, and Motion events. Corresponds to the %Y substitution for binding scripts.

---

**--sendevent boolean**

Boolean must be a boolean value; it specifies the send_event field for the event. Valid for all events. Corresponds to the %E substitution for binding scripts.

---

**--serial number**

Number must be an integer; it specifies the serial field for the event. Valid for all events. Corresponds to the %# substitution for binding scripts.

---

**--state state**

State specifies the state field for the event. For KeyPress, KeyRelease, ButtonPress, ButtonRelease, Enter, Leave, and Motion events it must be an integer value. For Visibility events it must be one of VisibilityUnobscured, VisibilityPartiallyObscured, or VisibilityFullyObscured. This option overrides any modifiers such as Meta or Control specified in the base event. Corresponds to the %s substitution for binding scripts.

---

**--subwindow window**

Window specifies the subwindow field for the event, either as a path name for a Tk widget or as an integer window identifier. Valid for KeyPress, KeyRelease, ButtonPress, ButtonRelease, Enter, Leave, and Motion events. Similar to %S substitution for binding scripts.

---

**--time integer**
Integer must be an integer value; it specifies the time field for the event. Valid for KeyPress, KeyRelease, ButtonPress, ButtonRelease, Enter, Leave, Motion, and Property events. Corresponds to the %t substitution for binding scripts.

--warp boolean
boolean must be a boolean value; it specifies whether the screen pointer should be warped as well. Valid for KeyPress, KeyRelease, ButtonPress, ButtonRelease, and Motion events. The pointer will only warp to a window if it is mapped.

--width size
Size must be a screen distance; it specifies the width field for the event. Valid for Configure events. Corresponds to the %w substitution for binding scripts.

--when when
When determines when the event will be processed; it must have one of the following values:

now
Process the event immediately, before the command returns. This also happens if the --when option is omitted.

tail
Place the event on Tcl's event queue behind any events already queued for this application.

head
Place the event at the front of Tcl's event queue, so that it will be handled before any other events already queued.

mark
Place the event at the front of Tcl's event queue but behind any other events already queued with --when mark. This option is useful when generating a series of events that should be processed in order but at the front of the queue.

--x coord
Coord must be a screen distance; it specifies the x field for the event. Valid for KeyPress, KeyRelease, ButtonPress, ButtonRelease, Motion, Enter, Leave, Expose, Configure, Gravity, and Reparent events. Corresponds to the %x substitution for binding scripts. If Window is empty the coordinate is relative to the screen, and this option corresponds to the %X substitution for binding scripts.

--y coord
Coord must be a screen distance; it specifies the y field for the event. Valid for KeyPress, KeyRelease, ButtonPress, ButtonRelease, Motion, Enter, Leave, Expose, Configure, Gravity, and Reparent events. Corresponds to the %y substitution for binding scripts. If Window is empty the coordinate is relative to the screen, and this option corresponds to the %Y substitution for binding scripts.

Any options that are not specified when generating an event are filled with the value 0, except for serial, which is filled with the next X event serial number.

VIRTUAL EVENT EXAMPLES

In order for a virtual event binding to trigger, two things must happen. First, the virtual event must be defined with the event add command. Second, a binding must be created for the virtual event with the bind command.
command. Consider the following virtual event definitions:

```
event add <<Paste>> <Control-y>
event add <<Paste>> <Button-2>
event add <<Save>> <Control-X><Control-S>
event add <<Save>> <Shift-F12>
```

In the `bind` command, a virtual event can be bound like any other built-in event type as follows:

```
bind Entry <<Paste>> {%W insert [selection get]}
```

The double angle brackets are used to specify that a virtual event is being bound. If the user types Control–y or presses button 2, or if a `<<Paste>>` virtual event is synthesized with `event generate`, then the `<<Paste>>` binding will be invoked.

If a virtual binding has the exact same sequence as a separate physical binding, then the physical binding will take precedence. Consider the following example:

```
event add <<Paste>> <Control-y> <Meta-Control-y>
binding Entry <Control-y> {puts Control-y}
binding Entry <<Paste>> {puts Paste}
```

When the user types Control–y the `<Control-y>` binding will be invoked, because a physical event is considered more specific than a virtual event, all other things being equal. However, when the user types Meta–Control–y the `<<Paste>>` binding will be invoked, because the `Meta` modifier in the physical pattern associated with the virtual binding is more specific than the `<Control-y>` sequence for the physical event.

Bindings on a virtual event may be created before the virtual event exists. Indeed, the virtual event never actually needs to be defined, for instance, on platforms where the specific virtual event would meaningless or ungeneratable.

When a definition of a virtual event changes at run time, all windows will respond immediately to the new definition. Starting from the preceding example, if the following code is executed:

```
binding <Entry> <Control-y> {}
event add <<Paste>> <Key-F6>
```

the behavior will change such in two ways. First, the shadowed `<<Paste>>` binding will emerge. Typing Control–y will no longer invoke the `<Control-y>` binding, but instead invoke the virtual event `<<Paste>>`. Second, pressing the F6 key will now also invoke the `<<Paste>>` binding.

**SEE ALSO**

`bind`
KEYWORDS

event, binding, define, handle, virtual event

Copyright ' 1996 Sun Microsystems, Inc.
Copyright ' 1998-2000 Ajuba Solutions.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
The `focus` command is used to manage the Tk input focus. At any given time, one window on each display is designated as the `focus window`; any key press or key release events for the display are sent to that window. It is normally up to the window manager to redirect the focus among the top–level windows of a display. For example, some window managers automatically set the input focus to a top–level window whenever the mouse enters it; others redirect the input focus only when the user clicks on a window. Usually the window manager will set the focus only to top–level windows, leaving it up to the application to redirect the focus among the children of the top–level.

Tk remembers one focus window for each top–level (the most recent descendant of that top–level to receive the focus); when the window manager gives the focus to a top–level, Tk automatically redirects it to the remembered window. Within a top–level Tk uses an *explicit* focus model by default. Moving the mouse within a top–level does not normally change the focus; the focus changes only when a widget decides...
explicitly to claim the focus (e.g., because of a button click), or when the user types a key such as Tab that
moves the focus.

The Tcl procedure `tk_focusFollowsMouse` may be invoked to create an *implicit* focus model: it reconfigures
Tk so that the focus is set to a window whenever the mouse enters it. The Tcl procedures `tk_focusNext` and
`tk_focusPrev` implement a focus order among the windows of a top–level; they are used in the default
bindings for Tab and Shift–Tab, among other things.

The `focus` command can take any of the following forms:

**Focus**

Returns the path name of the focus window on the display containing the application's main window,
or an empty string if no window in this application has the focus on that display. Note: it is better to
specify the display explicitly using `-displayof` (see below) so that the code will work in applications
using multiple displays.

**Focus window**

If the application currently has the input focus on `window`'s display, this command resets the input
focus for `window`'s display to `window` and returns an empty string. If the application doesn't currently
have the input focus on `window`'s display, `window` will be remembered as the focus for its top–level;
the next time the focus arrives at the top–level, Tk will redirect it to `window`. If `window` is an empty
string then the command does nothing.

**Focus –displayof window**

Returns the name of the focus window on the display containing `window`. If the focus window for
`window`'s display isn't in this application, the return value is an empty string.

**Focus –force window**

Sets the focus of `window`'s display to `window`, even if the application doesn't currently have the input
focus for the display. This command should be used sparingly, if at all. In normal usage, an
application should not claim the focus for itself; instead, it should wait for the window manager to
give it the focus. If `window` is an empty string then the command does nothing.

**Focus –lastfor window**

Returns the name of the most recent window to have the input focus among all the windows in the
same top–level as `window`. If no window in that top–level has ever had the input focus, or if the most
recent focus window has been deleted, then the name of the top–level is returned. The return value is
the window that will receive the input focus the next time the window manager gives the focus to the
top–level.

**Quirks**

When an internal window receives the input focus, Tk doesn't actually set the X focus to that window; as far
as X is concerned, the focus will stay on the top–level window containing the window with the focus.
However, Tk generates FocusIn and FocusOut events just as if the X focus were on the internal window. This
approach gets around a number of problems that would occur if the X focus were actually moved; the fact that
the X focus is on the top–level is invisible unless you use C code to query the X server directly.
EXAMPLE

To make a window that only participates in the focus traversal ring when a variable is set, add the following bindings to the widgets before and after it in that focus ring:

```tcl
button .before -text "Before"
button .middle -text "Middle"
button .after -text "After"
checkbutton .flag -variable traverseToMiddle -takefocus 0
pack .flag -side left
pack .before .middle .after
bind .before <Tab> { 
    if {!$traverseToMiddle} {
        focus .after
        break
    }
}
bind .after <Shift-Tab> { 
    if {!$traverseToMiddle} {
        focus .before
        break
    }
}
focus .before
```

KEYWORDS

events, focus, keyboard, top-level, window manager

Copyright ' 1990-1994 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

font – Create and inspect fonts.

SYNOPSIS

    font option ?arg arg ...?

DESCRIPTION

    font actual font ?−displayof window? ?option?
    font configure fontname ?option? ?value option value ...?
    font create ?fontname? ?option value ...?
    font delete fontname ?fontname ...?
    font families ?−displayof window?
    font measure font ?−displayof window? text
    font metrics font ?−displayof window? ?option?
    font names

FONT DESCRIPTION

[1] fontname
[2] systemfont
[4] X–font names (XLFD)
[5] option value ?option value ...?

FONT METRICS

−ascent
−descent
−linespace
−fixed

FONT OPTIONS

−family name
−size size
−weight weight
−slant slant
−underline boolean
−overstrike boolean

PLATFORM–SPECIFIC ISSUES

X Windows:
MS Windows:
Macintosh:

EXAMPLE

SEE ALSO

KEYWORDS
NAME

font – Create and inspect fonts.

SYNOPSIS

font option ?arg arg ...?

DESCRIPTION

The font command provides several facilities for dealing with fonts, such as defining named fonts and inspecting the actual attributes of a font. The command has several different forms, determined by the first argument. The following forms are currently supported:

font actual font ?–displayof window? ?option?
Returns information about the actual attributes that are obtained when font is used on window's display; the actual attributes obtained may differ from the attributes requested due to platform-dependant limitations, such as the availability of font families and pointsizes. font is a font description; see FONT DESCRIPTIONS below. If the window argument is omitted, it defaults to the main window. If option is specified, returns the value of that attribute; if it is omitted, the return value is a list of all the attributes and their values. See FONT OPTIONS below for a list of the possible attributes.

font configure fontname ?option? ?value option value ...?
Query or modify the desired attributes for the named font called fontname. If no option is specified, returns a list describing all the options and their values for fontname. If a single option is specified with no value, then returns the current value of that attribute. If one or more option–value pairs are specified, then the command modifies the given named font to have the given values; in this case, all widgets using that font will redisplay themselves using the new attributes for the font. See FONT OPTIONS below for a list of the possible attributes.

font create ?fontname? ?option value ...?
Creates a new named font and returns its name. fontname specifies the name for the font; if it is omitted, then Tk generates a new name of the form fontx, where x is an integer. There may be any number of option–value pairs, which provide the desired attributes for the new named font. See FONT OPTIONS below for a list of the possible attributes.

font delete fontname ?fontname ...?
Delete the specified named fonts. If there are widgets using the named font, the named font won't actually be deleted until all the instances are released. Those widgets will continue to display using the last known values for the named font. If a deleted named font is subsequently recreated with another call to font create, the widgets will use the new named font and redisplay themselves using the new attributes of that font.

font families ?–displayof window?
The return value is a list of the case–insensitive names of all font families that exist on window's display. If the window argument is omitted, it defaults to the main window.

font measure font ?–displayof window? text
Measures the amount of space the string text would use in the given font when displayed in window.

font is a font description; see FONT DESCRIPTIONS below. If the window argument is omitted, it
defaults to the main window. The return value is the total width in pixels of text, not including the
extra pixels used by highly exaggerated characters such as cursive `"f". If the string contains newlines
or tabs, those characters are not expanded or treated specially when measuring the string.

**font metrics font ?−displayof window? ?option?**

Returns information about the metrics (the font−specific data), for font when it is used on window's
display. font is a font description; see FONT DESCRIPTIONS below. If the window argument is
omitted, it defaults to the main window. If option is specified, returns the value of that metric; if it is
omitted, the return value is a list of all the metrics and their values. See FONT METRICS below for a
list of the possible metrics.

**font names**

The return value is a list of all the named fonts that are currently defined.

### FONT DESCRIPTION

The following formats are accepted as a font description anywhere font is specified as an argument above;
these same forms are also permitted when specifying the −font option for widgets.

**[1] fontname**

The name of a named font, created using the font create command. When a widget uses a named
font, it is guaranteed that this will never cause an error, as long as the named font exists, no matter
what potentially invalid or meaningless set of attributes the named font has. If the named font cannot
be displayed with exactly the specified attributes, some other close font will be substituted
automatically.

**[2] systemfont**

The platform−specific name of a font, interpreted by the graphics server. This also includes, under X,
an XLFD (see [4]) for which a single `"*" character was used to elide more than one field in the
middle of the name. See PLATFORM−SPECIFIC issues for a list of the system fonts.


A properly formed list whose first element is the desired font family and whose optional second
element is the desired size. The interpretation of the size attribute follows the same rules described for
−size in FONT OPTIONS below. Any additional optional arguments following the size are font
styles. Possible values for the style arguments are as follows:

    normal bold roman italic underline overstrike

**[4] X−font names (XLFD)**

A Unix−centric font name of the form

−foundry−family−weight−slant−setwidth−addstyle−pixel−point−resx−resy−spacing−width−charset−encoding

The `"*" character may be used to skip individual fields that the user does not care about. There must
be exactly one `"*" for each field skipped, except that a `"*" at the end of the XLFD skips any
remaining fields; the shortest valid XLFD is simply `"*", signifying all fields as defaults. Any fields
that were skipped are given default values. For compatibility, an XLFD always chooses a font of the
specified pixel size (not point size); although this interpretation is not strictly correct, all existing
applications using XLFDs assumed that one `"point" was in fact one pixel and would display
incorrectly (generally larger) if the correct size font were actually used.

[5] option value ?option value ...?  
A properly formed list of option–value pairs that specify the desired attributes of the font, in the same format used when defining a named font; see FONT OPTIONS below.

When font description font is used, the system attempts to parse the description according to each of the above five rules, in the order specified. Cases [1] and [2] must match the name of an existing named font or of a system font. Cases [3], [4], and [5] are accepted on all platforms and the closest available font will be used. In some situations it may not be possible to find any close font (e.g., the font family was a garbage value); in that case, some system–dependant default font is chosen. If the font description does not match any of the above patterns, an error is generated.

FONT METRICS

The following options are used by the font metrics command to query font–specific data determined when the font was created. These properties are for the whole font itself and not for individual characters drawn in that font. In the following definitions, the ``baseline'' of a font is the horizontal line where the bottom of most letters line up; certain letters, such as lower–case ``g'' stick below the baseline.

−ascent  
The amount in pixels that the tallest letter sticks up above the baseline of the font, plus any extra blank space added by the designer of the font.

−descent  
The largest amount in pixels that any letter sticks down below the baseline of the font, plus any extra blank space added by the designer of the font.

−linespace  
Returns how far apart vertically in pixels two lines of text using the same font should be placed so that none of the characters in one line overlap any of the characters in the other line. This is generally the sum of the ascent above the baseline line plus the descent below the baseline.

−fixed  
Returns a boolean flag that is ``1'' if this is a fixed–width font, where each normal character is the same width as all the other characters, or is ``0'' if this is a proportionally–spaced font, where individual characters have different widths. The widths of control characters, tab characters, and other non–printing characters are not included when calculating this value.

FONT OPTIONS

The following options are supported on all platforms, and are used when constructing a named font or when specifying a font using style [5] as above:

−family name  
The case–insensitive font family name. Tk guarantees to support the font families named Courier (a monospaced ``typewriter'' font), Times (a serifed ``newspaper'' font), and Helvetica (a sans–serif ``European'' font). The most closely matching native font family will automatically be substituted when one of the above font families is used. The name may also be the name of a native,
platform-specific font family; in that case it will work as desired on one platform but may not display correctly on other platforms. If the family is unspecified or unrecognized, a platform-specific default font will be chosen.

```
--size size
```

The desired size of the font. If the `size` argument is a positive number, it is interpreted as a size in points. If `size` is a negative number, its absolute value is interpreted as a size in pixels. If a font cannot be displayed at the specified size, a nearby size will be chosen. If `size` is unspecified or zero, a platform-dependent default size will be chosen.

Sizes should normally be specified in points so the application will remain the same ruler size on the screen, even when changing screen resolutions or moving scripts across platforms. However, specifying pixels is useful in certain circumstances such as when a piece of text must line up with respect to a fixed-size bitmap. The mapping between points and pixels is set when the application starts, based on properties of the installed monitor, but it can be overridden by calling the `tk scaling` command.

```
--weight weight
```

The nominal thickness of the characters in the font. The value `normal` specifies a normal weight font, while `bold` specifies a bold font. The closest available weight to the one specified will be chosen. The default weight is `normal`.

```
--slant slant
```

The amount the characters in the font are slanted away from the vertical. Valid values for slant are `roman` and `italic`. A roman font is the normal, upright appearance of a font, while an italic font is one that is tilted some number of degrees from upright. The closest available slant to the one specified will be chosen. The default slant is `roman`.

```
--underline boolean
```

The value is a boolean flag that specifies whether characters in this font should be underlined. The default value for underline is `false`.

```
--overstrike boolean
```

The value is a boolean flag that specifies whether a horizontal line should be drawn through the middle of characters in this font. The default value for overstrike is `false`.

### PLATFORM–SPECIFIC ISSUES

The following named system fonts are supported:

**X Windows:**

All valid X font names, including those listed by `xlsfonts(1)`, are available.

**MS Windows:**

- `system`
- `ansi`
- `device`
- `systemfixed`
- `ansifixed`
- `oemfixed`

**Macintosh:**

- `system`
- `application`
EXAMPLE

Fill a text widget with lots of font demonstrators, one for every font family installed on your system:

```tcl
pack [text .t -wrap none] -fill both -expand 1
set count 0
set tabwidth 0
foreach family [lsort -dictionary [font families]] {
    .t tag configure f[increment count] -font [list $family 10]
    .t insert end ${family}:	 {} \n        "This is a simple sampler\n" f$count
    set w [font measure [.t cget -font] ${family}:
    if {$w+5 > $tabwidth} {
        set tabwidth [expr {$w+5}]
        .t configure -tabs $tabwidth
    }
}
```

SEE ALSO

options

KEYWORDS

font

---

Copyright © 1996 Sun Microsystems, Inc.
NAME
frame – Create and manipulate frame widgets

SYNOPSIS
frame pathName ?options?

STANDARD OPTIONS
-`borderwidth` or `-bd, borderWidth, BorderWidth`
-`cursor, cursor, Cursor`
-`highlightbackground, highlightBackground, HighlightBackground`
-`highlightcolor, highlightColor, HighlightColor`
-`highlightthickness, highlightThickness, HighlightThickness`
-`padX, padX, Pad`
-`padY, padY, Pad`
-`relief, relief, Relief`
-`takefocus, takeFocus, TakeFocus`

WIDGET–SPECIFIC OPTIONS
-`background, background, Background`
-`class, class, Class`
-`colormap, colormap, Colormap`
-`container, container, Container`
-`height, height, Height`
-`visual, visual, Visual`
-`width, width, Width`

DESCRIPTION

WIDGET COMMAND
`pathName cget option`
`pathName configure ?option? ?value option value ...?`

BINDINGS
SEE ALSO
KEYWORDS
STANDARD OPTIONS

−borderwidth or −bd, borderWidth, BorderWidth
−cursor, cursor, Cursor
−highlightbackground, highlightBackground, HighlightBackground
−highlightcolor, highlightColor, HighlightColor
−highlightthickness, highlightThickness, HighlightThickness
−padx, padX, Pad
−pady, padY, Pad
−relief, relief, Relief
−takefocus, takeFocus, TakeFocus

WIDGET−SPECIFIC OPTIONS

Command−Line Name: −background
Database Name: background
Database Class: Background

This option is the same as the standard background option except that its value may also be specified as an empty string. In this case, the widget will display no background or border, and no colors will be consumed from its colormap for its background and border.

Command−Line Name: −class
Database Name: class
Database Class: Class

Specifies a class for the window. This class will be used when querying the option database for the window's other options, and it will also be used later for other purposes such as bindings. The class option may not be changed with the configure widget command.

Command−Line Name: −colormap
Database Name: colormap
Database Class: Colormap

Specifies a colormap to use for the window. The value may be either new, in which case a new colormap is created for the window and its children, or the name of another window (which must be on the same screen and have the same visual as pathName), in which case the new window will use the colormap from the specified window. If the colormap option is not specified, the new window uses the same colormap as its parent. This option may not be changed with the configure widget command.

Command−Line Name: −container
Database Name: container
Database Class: Container

The value must be a boolean. If true, it means that this window will be used as a container in which some other application will be embedded (for example, a Tk toplevel can be embedded using the −use option). The window will support the appropriate window manager protocols for things like geometry requests. The window should not have any children of its own in this application. This option may not be changed with the configure widget command.

Command−Line Name: −height
Database Name: height
*Database Class: Height*

Specifies the desired height for the window in any of the forms acceptable to `Tk_GetPixels`. If this option is less than or equal to zero then the window will not request any size at all.

*Command-Line Name:* −visual

*Database Name:* visual

*Database Class:* Visual

Specifies visual information for the new window in any of the forms accepted by `Tk_GetVisual`. If this option is not specified, the new window will use the same visual as its parent. The `visual` option may not be modified with the `configure` widget command.

*Command-Line Name:* −width

*Database Name:* width

*Database Class:* Width

Specifies the desired width for the window in any of the forms acceptable to `Tk_GetPixels`. If this option is less than or equal to zero then the window will not request any size at all.

### DESCRIPTION

The `frame` command creates a new window (given by the `pathName` argument) and makes it into a frame widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the frame such as its background color and relief. The `frame` command returns the path name of the new window.

A frame is a simple widget. Its primary purpose is to act as a spacer or container for complex window layouts. The only features of a frame are its background color and an optional 3-D border to make the frame appear raised or sunken.

### WIDGET COMMAND

The `frame` command creates a new Tcl command whose name is the same as the path name of the frame's window. This command may be used to invoke various operations on the widget. It has the following general form:

```tcl
pathName option ?arg arg ...?
```

*PathName* is the name of the command, which is the same as the frame widget's path name. *Option* and the *args* determine the exact behavior of the command. The following commands are possible for frame widgets:

```tcl
pathName cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the `frame` command.
```

```tcl
pathName configure ?option? ?value option value ...?

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see `Tk_ConfigureInfo` for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no
```
option is specified). If one or more option–value pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the *frame* command.

**BINDINGS**

When a new frame is created, it has no default event bindings: frames are not intended to be interactive.

**SEE ALSO**

`labelframe`, `toplevel`

**KEYWORDS**

`frame`, `widget`
NAME

grab – Confine pointer and keyboard events to a window sub–tree

SYNOPSIS

grab ?–global? window
grab option ?arg arg ...?

DESCRIPTION

This command implements simple pointer and keyboard grabs for Tk. Tk's grabs are different than the grabs
described in the Xlib documentation. When a grab is set for a particular window, Tk restricts all pointer events
to the grab window and its descendants in Tk's window hierarchy. Whenever the pointer is within the grab
window's subtree, the pointer will behave exactly the same as if there had been no grab at all and all events
will be reported in the normal fashion. When the pointer is outside window's tree, button presses and releases
and mouse motion events are reported to window, and window entry and window exit events are ignored. The
grab subtree ``owns'' the pointer: windows outside the grab subtree will be visible on the screen but they will
be insensitive until the grab is released. The tree of windows underneath the grab window can include
top–level windows, in which case all of those top–level windows and their descendants will continue to
receive mouse events during the grab.

Two forms of grabs are possible: local and global. A local grab affects only the grabbing application: events
will be reported to other applications as if the grab had never occurred. Grabs are local by default. A global
grab locks out all applications on the screen, so that only the given subtree of the grabbing application will be
sensitive to pointer events (mouse button presses, mouse button releases, pointer motions, window entries,
and window exits). During global grabs the window manager will not receive pointer events either.

During local grabs, keyboard events (key presses and key releases) are delivered as usual: the window
manager controls which application receives keyboard events, and if they are sent to any window in the
grabbing application then they are redirected to the focus window. During a global grab Tk grabs the
keyboard so that all keyboard events are always sent to the grabbing application. The focus command is still
used to determine which window in the application receives the keyboard events. The keyboard grab is
released when the grab is released.

Grabs apply to particular displays. If an application has windows on multiple displays then it can establish a
separate grab on each display. The grab on a particular display affects only the windows on that display. It is
possible for different applications on a single display to have simultaneous local grabs, but only one
application can have a global grab on a given display at once.

The grab command can take any of the following forms:

```
grab ?–global? window
   Same as grab set, described below.

grab current ?window?
   If window is specified, returns the name of the current grab window in this application for window's
display, or an empty string if there is no such window. If window is omitted, the command returns a
list whose elements are all of the windows grabbed by this application for all displays, or an empty
string if the application has no grabs.

grab release window
   Releases the grab on window if there is one, otherwise does nothing. Returns an empty string.

grab set ?–global? window
   Sets a grab on window. If –global is specified then the grab is global, otherwise it is local. If a grab
   was already in effect for this application on window's display then it is automatically released. If there
   is already a grab on window and it has the same global/local form as the requested grab, then the
   command does nothing. Returns an empty string.

grab status window
   Returns none if no grab is currently set on window, local if a local grab is set on window, and global
   if a global grab is set.
```

**WARNING**

It is very easy to use global grabs to render a display completely unusable (e.g. by setting a grab on a widget
which does not respond to events and not providing any mechanism for releasing the grab). Take extreme care
when using them!
BUGS

It took an incredibly complex and gross implementation to produce the simple grab effect described above. Given the current implementation, it isn't safe for applications to use the Xlib grab facilities at all except through the Tk grab procedures. If applications try to manipulate X's grab mechanisms directly, things will probably break.

If a single process is managing several different Tk applications, only one of those applications can have a local grab for a given display at any given time. If the applications are in different processes, this restriction doesn't exist.

EXAMPLE

Set a grab so that only one button may be clicked out of a group. The other buttons are unresponsive to the mouse until the middle button is clicked.

pack [button .b1 -text "Click me! #1" -command {destroy .b1}]
pack [button .b2 -text "Click me! #2" -command {destroy .b2}]
pack [button .b3 -text "Click me! #3" -command {destroy .b3}]
grab .b2

KEYWORDS

grab, keyboard events, pointer events, window

Copyright ' 1992 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

grid – Geometry manager that arranges widgets in a grid

SYNOPSIS

grid option arg ?arg ...?

description

grid slave ?slave ... ?options?
grid bbox master ?column row? ?column2 row2?
grid columnconfigure master index ?–option value...?
grid configure slave ?slave ... ?options?
    –column n
    –columnspan n
    –in other
    –ipadx amount
    –ipady amount
    –padx amount
    –pady amount
    –row n
    –rowspan n
    –sticky style
grid forget slave ?slave ...?
grid info slave
grid location master x y
grid propagate master ?boolean?
grid rowconfigure master index ?–option value...?
grid remove slave ?slave ...?
grid size master
grid slaves master ?–option value?

RELATIVE PLACEMENT

=  
  
^  

THE GRID ALGORITHM

GEOMETRY PROPAGATION

RESTRICTIONS ON MASTER WINDOWS

STACKING ORDER

CREDITS

EXAMPLES

SEE ALSO

KEYWORDS
NAME

grid – Geometry manager that arranges widgets in a grid

SYNOPSIS

grid option arg ?arg ...?

DESCRIPTION

The grid command is used to communicate with the grid geometry manager that arranges widgets in rows and columns inside of another window, called the geometry master (or master window). The grid command can have any of several forms, depending on the option argument:

grid slave ?slave ...? ?options?
    If the first argument to grid is suitable as the first slave argument to grid configure, either a window name (any value starting with .) or one of the characters x or ^ (see the RELATIVE PLACEMENT section below), then the command is processed in the same way as grid configure.

grid bbox master ?column row? ?column2 row2?
    With no arguments, the bounding box (in pixels) of the grid is returned. The return value consists of 4 integers. The first two are the pixel offset from the master window (x then y) of the top–left corner of the grid, and the second two integers are the width and height of the grid, also in pixels. If a single column and row is specified on the command line, then the bounding box for that cell is returned, where the top left cell is numbered from zero. If both column and row arguments are specified, then the bounding box spanning the rows and columns indicated is returned.

grid columnconfigure master index ?−option value...?
    Query or set the column properties of the index column of the geometry master, master. The valid options are −minsize, −weight, −uniform and −pad. If one or more options are provided, then index may be given as a list of column indices to which the configuration options will operate on. The −minsize option sets the minimum size, in screen units, that will be permitted for this column. The −weight option (an integer value) sets the relative weight for apportioning any extra spaces among columns. A weight of zero (0) indicates the column will not deviate from its requested size. A column whose weight is two will grow at twice the rate as a column of weight one when extra space is allocated to the layout. The −uniform option, when a non–empty value is supplied, places the column in a uniform group with other columns that have the same value for −uniform. The space for columns belonging to a uniform group is allocated so that their sizes are always in strict proportion to their −weight values. See THE GRID ALGORITHM below for further details. The −pad option specifies the number of screen units that will be added to the largest window contained completely in that column when the grid geometry manager requests a size from the containing window. If only an option is specified, with no value, the current value of that option is returned. If only the master window and index is specified, all the current settings are returned in a list of "−option value" pairs.

grid configure slave ?slave ...? ?options?
    The arguments consist of the names of one or more slave windows followed by pairs of arguments that specify how to manage the slaves. The characters −, x and ^, can be specified instead of a window name to alter the default location of a slave, as described in the RELATIVE PLACEMENT
section, below. The following options are supported:

- **column n**
  Insert the slave so that it occupies the n-th column in the grid. Column numbers start with 0. If this option is not supplied, then the slave is arranged just to the right of previous slave specified on this call to grid, or column "0" if it is the first slave. For each x that immediately precedes the slave, the column position is incremented by one. Thus the x represents a blank column for this row in the grid.

- **columnspan n**
  Insert the slave so that it occupies n columns in the grid. The default is one column, unless the window name is followed by a -, in which case the columnspan is incremented once for each immediately following -.

- **in other**
  Insert the slave(s) in the master window given by other. The default is the first slave's parent window.

- **ipadx amount**
  The amount specifies how much horizontal internal padding to leave on each side of the slave(s). This space is added inside the slave(s) border. The amount must be a valid screen distance, such as 2 or .5c. It defaults to 0.

- **ipady amount**
  The amount specifies how much vertical internal padding to leave on the top and bottom of the slave(s). This space is added inside the slave(s) border. The amount defaults to 0.

- **padx amount**
  The amount specifies how much horizontal external padding to leave on each side of the slave(s), in screen units. Amount may be a list of two values to specify padding for left and right separately. The amount defaults to 0. This space is added outside the slave(s) border.

- **pady amount**
  The amount specifies how much vertical external padding to leave on the top and bottom of the slave(s), in screen units. Amount may be a list of two values to specify padding for top and bottom separately. The amount defaults to 0. This space is added outside the slave(s) border.

- **row n**
  Insert the slave so that it occupies the n-th row in the grid. Row numbers start with 0. If this option is not supplied, then the slave is arranged on the same row as the previous slave specified on this call to grid, or the first unoccupied row if this is the first slave.

- **rowspan n**
  Insert the slave so that it occupies n rows in the grid. The default is one row. If the next grid command contains ^ characters instead of slaves that line up with the columns of this slave, then the rowspan of this slave is extended by one.

- **sticky style**
  If a slave's cell is larger than its requested dimensions, this option may be used to position (or stretch) the slave within its cell. Style is a string that contains zero or more of the characters n, s, e or w. The string can optionally contains spaces or commas, but they are ignored. Each letter refers to a side (north, south, east, or west) that the slave will "stick" to. If both n and s (or e and w) are specified, the slave will be stretched to fill the entire height (or width) of its cavity. The sticky option subsumes the combination of --anchor and --fill that is used by
pack. The default is {}, which causes the slave to be centered in its cavity, at its requested size.

If any of the slaves are already managed by the geometry manager then any unspecified options for them retain their previous values rather than receiving default values.

grid forget slave ?slave ...?
Removes each of the slaves from grid for its master and unmaps their windows. The slaves will no longer be managed by the grid geometry manager. The configuration options for that window are forgotten, so that if the slave is managed once more by the grid geometry manager, the initial default settings are used.

grid info slave
Returns a list whose elements are the current configuration state of the slave given by slave in the same option–value form that might be specified to grid configure. The first two elements of the list are `"−in master"' where master is the slave's master.

grid location master x y
Given x and y values in screen units relative to the master window, the column and row number at that x and y location is returned. For locations that are above or to the left of the grid, −1 is returned.

grid propagate master ?boolean?
If boolean has a true boolean value such as 1 or on then propagation is enabled for master, which must be a window name (see GEOMETRY PROPAGATION below). If boolean has a false boolean value then propagation is disabled for master. In either of these cases an empty string is returned. If boolean is omitted then the command returns 0 or 1 to indicate whether propagation is currently enabled for master. Propagation is enabled by default.

grid rowconfigure master index ?−option value...?
Query or set the row properties of the index row of the geometry master, master. The valid options are −minsize, −weight, −uniform and −pad. If one or more options are provided, then index may be given as a list of row indices to which the configuration options will operate on. The −minsize option sets the minimum size, in screen units, that will be permitted for this row. The −weight option (an integer value) sets the relative weight for apportioning any extra spaces among rows. A weight of zero (0) indicates the row will not deviate from its requested size. A row whose weight is two will grow at twice the rate as a row of weight one when extra space is allocated to the layout. The −uniform option, when a non-empty value is supplied, places the row in a uniform group with other rows that have the same value for −uniform. The space for rows belonging to a uniform group is allocated so that their sizes are always in strict proportion to their −weight values. See THE GRID ALGORITHM below for further details. The −pad option specifies the number of screen units that will be added to the largest window contained completely in that row when the grid geometry manager requests a size from the containing window. If only an option is specified, with no value, the current value of that option is returned. If only the master window and index is specified, all the current settings are returned in a list of "−option value" pairs.

grid remove slave ?slave ...?
Removes each of the slaves from grid for its master and unmaps their windows. The slaves will no longer be managed by the grid geometry manager. However, the configuration options for that window are remembered, so that if the slave is managed once more by the grid geometry manager, the previous values are retained.

grid size master
Returns the size of the grid (in columns then rows) for master. The size is determined either by the
slave occupying the largest row or column, or the largest column or row with a minsize, weight, or pad that is non-zero.

grid slaves master ?–option value?

If no options are supplied, a list of all of the slaves in master are returned, most recently manages first. Option can be either –row or –column which causes only the slaves in the row (or column) specified by value to be returned.

RELATIVE PLACEMENT

The grid command contains a limited set of capabilities that permit layouts to be created without specifying the row and column information for each slave. This permits slaves to be rearranged, added, or removed without the need to explicitly specify row and column information. When no column or row information is specified for a slave, default values are chosen for column, row, columnspan and rowspan at the time the slave is managed. The values are chosen based upon the current layout of the grid, the position of the slave relative to other slaves in the same grid command, and the presence of the characters −, x, and ^ in grid command where slave names are normally expected.

−

This increases the columnspan of the slave to the left. Several −’s in a row will successively increase the columnspan. A − may not follow a ^ or a x, nor may it be the first slave argument to grid configure.

x

This leaves an empty column between the slave on the left and the slave on the right.

^

This extends the rowspan of the slave above the ^’s in the grid. The number of ^’s in a row must match the number of columns spanned by the slave above it.

THE GRID ALGORITHM

The grid geometry manager lays out its slaves in three steps. In the first step, the minimum size needed to fit all of the slaves is computed, then (if propagation is turned on), a request is made of the master window to become that size. In the second step, the requested size is compared against the actual size of the master. If the sizes are different, then spaces is added to or taken away from the layout as needed. For the final step, each slave is positioned in its row(s) and column(s) based on the setting of its sticky flag.

To compute the minimum size of a layout, the grid geometry manager first looks at all slaves whose columnspan and rowspan values are one, and computes the nominal size of each row or column to be either the minsize for that row or column, or the sum of the padding plus the size of the largest slave, whichever is greater. After that the rows or columns in each uniform group adapt to each other. Then the slaves whose rowspans or columnspans are greater than one are examined. If a group of rows or columns need to be increased in size in order to accommodate these slaves, then extra space is added to each row or column in the group according to its weight. For each group whose weights are all zero, the additional space is apportioned equally.
When multiple rows or columns belong to a uniform group, the space allocated to them is always in proportion to their weights. (A weight of zero is considered to be 1.) In other words, a row or column configured with \(-\text{weight 1} - \text{uniform a}\) will have exactly the same size as any other row or column configured with \(-\text{weight 1} - \text{uniform a}\). A row or column configured with \(-\text{weight 2} - \text{uniform b}\) will be exactly twice as large as one that is configured with \(-\text{weight 1} - \text{uniform b}\).

More technically, each row or column in the group will have a size equal to \(k \times \text{weight}\) for some constant \(k\). The constant \(k\) is chosen so that no row or column becomes smaller than its minimum size. For example, if all rows or columns in a group have the same weight, then each row or column will have the same size as the largest row or column in the group.

For masters whose size is larger than the requested layout, the additional space is apportioned according to the row and column weights. If all of the weights are zero, the layout is centered within its master. For masters whose size is smaller than the requested layout, space is taken away from columns and rows according to their weights. However, once a column or row shrinks to its minsize, its weight is taken to be zero. If more space needs to be removed from a layout than would be permitted, as when all the rows or columns are at their minimum sizes, the layout is clipped on the bottom and right.

**GEOMETRY PROPAGATION**

The grid geometry manager normally computes how large a master must be to just exactly meet the needs of its slaves, and it sets the requested width and height of the master to these dimensions. This causes geometry information to propagate up through a window hierarchy to a top-level window so that the entire sub-tree sizes itself to fit the needs of the leaf windows. However, the `grid propagate` command may be used to turn off propagation for one or more masters. If propagation is disabled then grid will not set the requested width and height of the master window. This may be useful if, for example, you wish for a master window to have a fixed size that you specify.

**RESTRICTIONS ON MASTER WINDOWS**

The master for each slave must either be the slave's parent (the default) or a descendant of the slave's parent. This restriction is necessary to guarantee that the slave can be placed over any part of its master that is visible without danger of the slave being clipped by its parent. In addition, all slaves in one call to `grid` must have the same master.

**STACKING ORDER**

If the master for a slave is not its parent then you must make sure that the slave is higher in the stacking order than the master. Otherwise the master will obscure the slave and it will appear as if the slave hasn't been managed correctly. The easiest way to make sure the slave is higher than the master is to create the master window first: the most recently created window will be highest in the stacking order.
CREDITS

The grid command is based on ideas taken from the GridBag geometry manager written by Doug. Stein, and the blt_table geometry manager, written by George Howlett.

EXAMPLES

A toplevel window containing a text widget and two scrollbars:

```tcl
# Make the widgets
toplevel .t
text .t.txt -wrap none -xscroll {.t.h set} -yscroll {.t.v set}
scrollbar .t.v -orient vertical   -command {.t.txt xview}
scrollbar .t.h -orient horizontal -command {.t.txt xview}
# Lay them out
grid .t.txt .t.v -sticky nsew
grid .t.h        -sticky nsew
# Tell the text widget to take all the extra room
grid rowconfigure .t 0 -weight 1
grid columnconfigure .t 0 -weight 1
```

Three widgets of equal width, despite their different "natural" widths:

```tcl
button .b -text "Foo"
entry .e -variable foo
label .l -text "This is a fairly long piece of text"
grids .b .e .l -sticky ew
grid columnconfigure . {0 1 2} -uniform allTheSame
```

SEE ALSO

pack, place

KEYWORDS

tcl

---

Copyright © 1996 Sun Microsystems, Inc.
Copyright © 1995-1997 Roger E. Critchlow Jr.
image

NAME

image – Create and manipulate images

SYNOPSIS

image option ?arg arg ...?

DESCRIPTION

image create type ?name? ?option value ...?
image delete ?name name ...
image height name
image inuse name
image names
image type name
image types
image width name

BUILT-IN IMAGE TYPES

bitmap
photo

SEE ALSO

KEYWORDS
note that the image command will silently overwrite any procedure that may currently be defined by
the given name, so choose the name wisely. It is recommended to use a separate namespace for image
names (e.g., ::img::logo, ::img::large).

*image delete ?name name ...?*

Deletes each of the named images and returns an empty string. If there are instances of the images
displayed in widgets, the images won't actually be deleted until all of the instances are released.
However, the association between the instances and the image manager will be dropped. Existing
instances will retain their sizes but redisplay as empty areas. If a deleted image is recreated with
another call to *image create*, the existing instances will use the new image.

*image height name*

Returns a decimal string giving the height of image name in pixels.

*image inuse name*

Returns a boolean value indicating whether or not the image given by name is in use by any widgets.

*image names*

Returns a list containing the names of all existing images.

*image type name*

Returns the type of image name (the value of the type argument to *image create* when the image was
created).

*image types*

Returns a list whose elements are all of the valid image types (i.e., all of the values that may be
supplied for the type argument to *image create*).

*image width name*

Returns a decimal string giving the width of image name in pixels.

Additional operations (e.g. writing the image to a file) may be available as subcommands of the image
instance command. See the manual page for the particular image type for details.

**BUILT–IN IMAGE TYPES**

The following image types are defined by Tk so they will be available in any Tk application. Individual
applications or extensions may define additional types.

*bitmap*

Each pixel in the image displays a foreground color, a background color, or nothing. See the *bitmap*
manual entry for more information.

*photo*

Displays a variety of full–color images, using dithering to approximate colors on displays with
limited color capabilities. See the *photo* manual entry for more information.

**SEE ALSO**

*bitmap, options, photo*
KEYWORDS

height, image, types of images, width
keysyms

NAME

keysyms – keysyms recognized by Tk

DESCRIPTION

Tk recognizes many keysyms when specifying key bindings (e.g. `bind . <Key–keysym>`). The following list enumerates the keysyms that will be recognized by Tk. Note that not all keysyms will be valid on all platforms. For example, on Unix systems, the presence of a particular keysym is dependant on the configuration of the keyboard modifier map. This list shows keysyms along with their decimal and hexadecimal values.

<table>
<thead>
<tr>
<th>Keysym</th>
<th>Decimal</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>space</td>
<td></td>
<td>32</td>
</tr>
<tr>
<td>exclam</td>
<td></td>
<td>33</td>
</tr>
<tr>
<td>quotedbl</td>
<td></td>
<td>34</td>
</tr>
<tr>
<td>numbersign</td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>dollar</td>
<td></td>
<td>36</td>
</tr>
<tr>
<td>percent</td>
<td></td>
<td>37</td>
</tr>
<tr>
<td>ampersand</td>
<td></td>
<td>38</td>
</tr>
<tr>
<td>quoteright</td>
<td></td>
<td>39</td>
</tr>
<tr>
<td>parenleft</td>
<td></td>
<td>40</td>
</tr>
<tr>
<td>parenright</td>
<td></td>
<td>41</td>
</tr>
<tr>
<td>asterisk</td>
<td></td>
<td>42</td>
</tr>
<tr>
<td>plus</td>
<td></td>
<td>43</td>
</tr>
<tr>
<td>comma</td>
<td></td>
<td>44</td>
</tr>
<tr>
<td>minus</td>
<td></td>
<td>45</td>
</tr>
<tr>
<td>period</td>
<td></td>
<td>46</td>
</tr>
<tr>
<td>slash</td>
<td></td>
<td>47</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>49</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>51</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>52</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>53</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>54</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>55</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>56</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>57</td>
</tr>
<tr>
<td>colon</td>
<td></td>
<td>58</td>
</tr>
<tr>
<td>semicolon</td>
<td></td>
<td>59</td>
</tr>
<tr>
<td>less</td>
<td></td>
<td>60</td>
</tr>
<tr>
<td>equal</td>
<td></td>
<td>61</td>
</tr>
<tr>
<td>greater</td>
<td></td>
<td>62</td>
</tr>
<tr>
<td>question</td>
<td></td>
<td>63</td>
</tr>
<tr>
<td>at</td>
<td></td>
<td>64</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>65</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>66</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>67</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>68</td>
</tr>
<tr>
<td>Character</td>
<td>ASCII Code</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>0x0045</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>0x0046</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>0x0047</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>0x0048</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>0x0049</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>0x004a</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>0x004b</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>0x004c</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>0x004d</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0x004e</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>0x004f</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>0x0050</td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>0x0051</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>0x0052</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>0x0053</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>0x0054</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>0x0055</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>0x0056</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>0x0057</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>0x0058</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>0x0059</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>0x005a</td>
<td></td>
</tr>
<tr>
<td>bracketleft</td>
<td>0x005b</td>
<td></td>
</tr>
<tr>
<td>backslash</td>
<td>0x005c</td>
<td></td>
</tr>
<tr>
<td>bracketright</td>
<td>0x005d</td>
<td></td>
</tr>
<tr>
<td>asciicircum</td>
<td>0x005e</td>
<td></td>
</tr>
<tr>
<td>underscore</td>
<td>0x005f</td>
<td></td>
</tr>
<tr>
<td>quoteleft</td>
<td>0x0060</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>0x0061</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>0x0062</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>0x0063</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>0x0064</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>0x0065</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>0x0066</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>0x0067</td>
<td></td>
</tr>
<tr>
<td>h</td>
<td>0x0068</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>0x0069</td>
<td></td>
</tr>
<tr>
<td>j</td>
<td>0x006a</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>0x006b</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>0x006c</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>0x006d</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>0x006e</td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>0x006f</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>0x0070</td>
<td></td>
</tr>
<tr>
<td>q</td>
<td>0x0071</td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>0x0072</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>0x0073</td>
<td></td>
</tr>
<tr>
<td>t</td>
<td>0x0074</td>
<td></td>
</tr>
<tr>
<td>u</td>
<td>0x0075</td>
<td></td>
</tr>
<tr>
<td>v</td>
<td>0x0076</td>
<td></td>
</tr>
<tr>
<td>w</td>
<td>0x0077</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0x0078</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>0x0079</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0x007a</td>
<td></td>
</tr>
<tr>
<td>braceleft</td>
<td>0x007b</td>
<td></td>
</tr>
</tbody>
</table>

**keysyms**
<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>bar</td>
<td>124</td>
</tr>
<tr>
<td>braceright</td>
<td>125</td>
</tr>
<tr>
<td>asciitilde</td>
<td>126</td>
</tr>
<tr>
<td>nobreakspace</td>
<td>160</td>
</tr>
<tr>
<td>exclamdown</td>
<td>161</td>
</tr>
<tr>
<td>cent</td>
<td>162</td>
</tr>
<tr>
<td>sterling</td>
<td>163</td>
</tr>
<tr>
<td>currency</td>
<td>164</td>
</tr>
<tr>
<td>yen</td>
<td>165</td>
</tr>
<tr>
<td>brokenbar</td>
<td>166</td>
</tr>
<tr>
<td>section</td>
<td>167</td>
</tr>
<tr>
<td>diaeress</td>
<td>168</td>
</tr>
<tr>
<td>copyright</td>
<td>169</td>
</tr>
<tr>
<td>ordfeminine</td>
<td>170</td>
</tr>
<tr>
<td>guillemotleft</td>
<td>171</td>
</tr>
<tr>
<td>notsign</td>
<td>172</td>
</tr>
<tr>
<td>hyphen</td>
<td>173</td>
</tr>
<tr>
<td>registered</td>
<td>174</td>
</tr>
<tr>
<td>macron</td>
<td>175</td>
</tr>
<tr>
<td>degree</td>
<td>176</td>
</tr>
<tr>
<td>plusminus</td>
<td>177</td>
</tr>
<tr>
<td>twosuperior</td>
<td>178</td>
</tr>
<tr>
<td>threesuperior</td>
<td>179</td>
</tr>
<tr>
<td>acute</td>
<td>180</td>
</tr>
<tr>
<td>mu</td>
<td>181</td>
</tr>
<tr>
<td>paragraph</td>
<td>182</td>
</tr>
<tr>
<td>periodcentered</td>
<td>183</td>
</tr>
<tr>
<td>cedilla</td>
<td>184</td>
</tr>
<tr>
<td>onesuperior</td>
<td>185</td>
</tr>
<tr>
<td>masculine</td>
<td>186</td>
</tr>
<tr>
<td>guillemotright</td>
<td>187</td>
</tr>
<tr>
<td>onequarter</td>
<td>188</td>
</tr>
<tr>
<td>onehalf</td>
<td>189</td>
</tr>
<tr>
<td>threequarters</td>
<td>190</td>
</tr>
<tr>
<td>questiondown</td>
<td>191</td>
</tr>
<tr>
<td>Agrave</td>
<td>192</td>
</tr>
<tr>
<td>Aacute</td>
<td>193</td>
</tr>
<tr>
<td>Acircumflex</td>
<td>194</td>
</tr>
<tr>
<td>Atilde</td>
<td>195</td>
</tr>
<tr>
<td>Adiaeresis</td>
<td>196</td>
</tr>
<tr>
<td>Aring</td>
<td>197</td>
</tr>
<tr>
<td>AE</td>
<td>198</td>
</tr>
<tr>
<td>Ccedilla</td>
<td>199</td>
</tr>
<tr>
<td>Egrave</td>
<td>200</td>
</tr>
<tr>
<td>Eacute</td>
<td>201</td>
</tr>
<tr>
<td>Ecircumflex</td>
<td>202</td>
</tr>
<tr>
<td>Ediaeresis</td>
<td>203</td>
</tr>
<tr>
<td>Igrave</td>
<td>204</td>
</tr>
<tr>
<td>Iacute</td>
<td>205</td>
</tr>
<tr>
<td>Icircumflex</td>
<td>206</td>
</tr>
<tr>
<td>Idiaeresis</td>
<td>207</td>
</tr>
<tr>
<td>Eth</td>
<td>208</td>
</tr>
<tr>
<td>Ntilde</td>
<td>209</td>
</tr>
<tr>
<td>Ograve</td>
<td>210</td>
</tr>
<tr>
<td>Oacute</td>
<td>211</td>
</tr>
<tr>
<td>Key</td>
<td>Value</td>
</tr>
<tr>
<td>--------------</td>
<td>-------</td>
</tr>
<tr>
<td>ocircumflex</td>
<td>212</td>
</tr>
<tr>
<td>otilde</td>
<td>213</td>
</tr>
<tr>
<td>odiaeresis</td>
<td>214</td>
</tr>
<tr>
<td>multiply</td>
<td>215</td>
</tr>
<tr>
<td>oblique</td>
<td>216</td>
</tr>
<tr>
<td>ugrave</td>
<td>217</td>
</tr>
<tr>
<td>uacute</td>
<td>218</td>
</tr>
<tr>
<td>ucircumflex</td>
<td>219</td>
</tr>
<tr>
<td>udiaeresis</td>
<td>220</td>
</tr>
<tr>
<td>yacute</td>
<td>221</td>
</tr>
<tr>
<td>thorn</td>
<td>222</td>
</tr>
<tr>
<td>ssharp</td>
<td>223</td>
</tr>
<tr>
<td>agrave</td>
<td>224</td>
</tr>
<tr>
<td>aacute</td>
<td>225</td>
</tr>
<tr>
<td>acircumflex</td>
<td>226</td>
</tr>
<tr>
<td>atilde</td>
<td>227</td>
</tr>
<tr>
<td>adiaeresis</td>
<td>228</td>
</tr>
<tr>
<td>aring</td>
<td>229</td>
</tr>
<tr>
<td>ae</td>
<td>230</td>
</tr>
<tr>
<td>ccedilla</td>
<td>231</td>
</tr>
<tr>
<td>egrave</td>
<td>232</td>
</tr>
<tr>
<td>eacute</td>
<td>233</td>
</tr>
<tr>
<td>ecircumflex</td>
<td>234</td>
</tr>
<tr>
<td>ediaeresis</td>
<td>235</td>
</tr>
<tr>
<td>igrave</td>
<td>236</td>
</tr>
<tr>
<td>iacute</td>
<td>237</td>
</tr>
<tr>
<td>icircumflex</td>
<td>238</td>
</tr>
<tr>
<td>idiaeresis</td>
<td>239</td>
</tr>
<tr>
<td>eth</td>
<td>240</td>
</tr>
<tr>
<td>ntilde</td>
<td>241</td>
</tr>
<tr>
<td>ograve</td>
<td>242</td>
</tr>
<tr>
<td>oacute</td>
<td>243</td>
</tr>
<tr>
<td>ocircumflex</td>
<td>244</td>
</tr>
<tr>
<td>otilde</td>
<td>245</td>
</tr>
<tr>
<td>odiaeresis</td>
<td>246</td>
</tr>
<tr>
<td>division</td>
<td>247</td>
</tr>
<tr>
<td>oslash</td>
<td>248</td>
</tr>
<tr>
<td>ugrave</td>
<td>249</td>
</tr>
<tr>
<td>uacute</td>
<td>250</td>
</tr>
<tr>
<td>ucircumflex</td>
<td>251</td>
</tr>
<tr>
<td>udiaeresis</td>
<td>252</td>
</tr>
<tr>
<td>yacute</td>
<td>253</td>
</tr>
<tr>
<td>thorn</td>
<td>254</td>
</tr>
<tr>
<td>ydiaeresis</td>
<td>255</td>
</tr>
<tr>
<td>aogonek</td>
<td>417</td>
</tr>
<tr>
<td>breve</td>
<td>418</td>
</tr>
<tr>
<td>lstroke</td>
<td>419</td>
</tr>
<tr>
<td>lcaron</td>
<td>421</td>
</tr>
<tr>
<td>sacute</td>
<td>422</td>
</tr>
<tr>
<td>scaron</td>
<td>425</td>
</tr>
<tr>
<td>scedilla</td>
<td>426</td>
</tr>
<tr>
<td>tcaron</td>
<td>427</td>
</tr>
<tr>
<td>zacute</td>
<td>428</td>
</tr>
<tr>
<td>zcaron</td>
<td>430</td>
</tr>
<tr>
<td>Key</td>
<td>Code</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
</tr>
<tr>
<td>Zabovedot</td>
<td>431</td>
</tr>
<tr>
<td>aogonek</td>
<td>433</td>
</tr>
<tr>
<td>ogonek</td>
<td>434</td>
</tr>
<tr>
<td>lstroke</td>
<td>435</td>
</tr>
<tr>
<td>lcaron</td>
<td>437</td>
</tr>
<tr>
<td>sacute</td>
<td>438</td>
</tr>
<tr>
<td>caron</td>
<td>439</td>
</tr>
<tr>
<td>scaron</td>
<td>441</td>
</tr>
<tr>
<td>scedilla</td>
<td>442</td>
</tr>
<tr>
<td>tcaron</td>
<td>443</td>
</tr>
<tr>
<td>zacute</td>
<td>444</td>
</tr>
<tr>
<td>doubleacute</td>
<td>445</td>
</tr>
<tr>
<td>zcaron</td>
<td>446</td>
</tr>
<tr>
<td>zabovedot</td>
<td>447</td>
</tr>
<tr>
<td>Racute</td>
<td>448</td>
</tr>
<tr>
<td>Abreve</td>
<td>451</td>
</tr>
<tr>
<td>Cacute</td>
<td>454</td>
</tr>
<tr>
<td>Ccaron</td>
<td>456</td>
</tr>
<tr>
<td>Ecune</td>
<td>458</td>
</tr>
<tr>
<td>Ccaron</td>
<td>459</td>
</tr>
<tr>
<td>Nacute</td>
<td>465</td>
</tr>
<tr>
<td>Ncaron</td>
<td>466</td>
</tr>
<tr>
<td>Oacute</td>
<td>469</td>
</tr>
<tr>
<td>Rcaron</td>
<td>472</td>
</tr>
<tr>
<td>Uring</td>
<td>473</td>
</tr>
<tr>
<td>Uddoubleacute</td>
<td>475</td>
</tr>
<tr>
<td>Tcedilla</td>
<td>478</td>
</tr>
<tr>
<td>Racute</td>
<td>480</td>
</tr>
<tr>
<td>Oacute</td>
<td>483</td>
</tr>
<tr>
<td>Cacute</td>
<td>486</td>
</tr>
<tr>
<td>Ccaron</td>
<td>488</td>
</tr>
<tr>
<td>Eacute</td>
<td>490</td>
</tr>
<tr>
<td>Oacute</td>
<td>492</td>
</tr>
<tr>
<td>Dcaron</td>
<td>495</td>
</tr>
<tr>
<td>Oacute</td>
<td>497</td>
</tr>
<tr>
<td>Ncaron</td>
<td>498</td>
</tr>
<tr>
<td>Oacute</td>
<td>501</td>
</tr>
<tr>
<td>Rcaron</td>
<td>504</td>
</tr>
<tr>
<td>Uring</td>
<td>505</td>
</tr>
<tr>
<td>Uddoubleacute</td>
<td>507</td>
</tr>
<tr>
<td>Tcedilla</td>
<td>510</td>
</tr>
<tr>
<td>Abovedot</td>
<td>511</td>
</tr>
<tr>
<td>Hstroke</td>
<td>673</td>
</tr>
<tr>
<td>Hcircumflex</td>
<td>678</td>
</tr>
<tr>
<td>Abovedot</td>
<td>681</td>
</tr>
<tr>
<td>Gbreve</td>
<td>683</td>
</tr>
<tr>
<td>Jcircumflex</td>
<td>684</td>
</tr>
<tr>
<td>Hstroke</td>
<td>689</td>
</tr>
<tr>
<td>Hcircumflex</td>
<td>694</td>
</tr>
<tr>
<td>Idotless</td>
<td>697</td>
</tr>
<tr>
<td>Gbreve</td>
<td>699</td>
</tr>
<tr>
<td>Jcircumflex</td>
<td>700</td>
</tr>
<tr>
<td>Cabovedot</td>
<td>709</td>
</tr>
<tr>
<td>Ccircumflex</td>
<td>710</td>
</tr>
<tr>
<td>Key</td>
<td>Code</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
</tr>
<tr>
<td>Gabovedot</td>
<td>725</td>
</tr>
<tr>
<td>Gcircumflex</td>
<td>728</td>
</tr>
<tr>
<td>Ubreve</td>
<td>733</td>
</tr>
<tr>
<td>Scircumflex</td>
<td>734</td>
</tr>
<tr>
<td>cabovedot</td>
<td>741</td>
</tr>
<tr>
<td>ccircumflex</td>
<td>742</td>
</tr>
<tr>
<td>gabovedot</td>
<td>757</td>
</tr>
<tr>
<td>gcircumflex</td>
<td>760</td>
</tr>
<tr>
<td>ubreve</td>
<td>765</td>
</tr>
<tr>
<td>scircumflex</td>
<td>766</td>
</tr>
<tr>
<td>kappa</td>
<td>930</td>
</tr>
<tr>
<td>Rcedilla</td>
<td>931</td>
</tr>
<tr>
<td>Itilde</td>
<td>933</td>
</tr>
<tr>
<td>Lcedilla</td>
<td>934</td>
</tr>
<tr>
<td>Emacron</td>
<td>938</td>
</tr>
<tr>
<td>Gcedilla</td>
<td>939</td>
</tr>
<tr>
<td>Tslash</td>
<td>940</td>
</tr>
<tr>
<td>rcedilla</td>
<td>947</td>
</tr>
<tr>
<td>itilde</td>
<td>949</td>
</tr>
<tr>
<td>lcedilla</td>
<td>950</td>
</tr>
<tr>
<td>emacron</td>
<td>954</td>
</tr>
<tr>
<td>gacute</td>
<td>955</td>
</tr>
<tr>
<td>tslash</td>
<td>956</td>
</tr>
<tr>
<td>ENG</td>
<td>957</td>
</tr>
<tr>
<td>eng</td>
<td>959</td>
</tr>
<tr>
<td>Amacron</td>
<td>960</td>
</tr>
<tr>
<td>Iogonek</td>
<td>967</td>
</tr>
<tr>
<td>Eabovedot</td>
<td>972</td>
</tr>
<tr>
<td>Imacron</td>
<td>975</td>
</tr>
<tr>
<td>Ncedilla</td>
<td>977</td>
</tr>
<tr>
<td>Omacron</td>
<td>978</td>
</tr>
<tr>
<td>Kcedilla</td>
<td>979</td>
</tr>
<tr>
<td>Uogonek</td>
<td>985</td>
</tr>
<tr>
<td>Utilde</td>
<td>989</td>
</tr>
<tr>
<td>Umacron</td>
<td>990</td>
</tr>
<tr>
<td>amacron</td>
<td>992</td>
</tr>
<tr>
<td>iogonek</td>
<td>999</td>
</tr>
<tr>
<td>eabovedot</td>
<td>1004</td>
</tr>
<tr>
<td>imacron</td>
<td>1007</td>
</tr>
<tr>
<td>ncedilla</td>
<td>1009</td>
</tr>
<tr>
<td>omacron</td>
<td>1010</td>
</tr>
<tr>
<td>kcedilla</td>
<td>1011</td>
</tr>
<tr>
<td>uogonek</td>
<td>1017</td>
</tr>
<tr>
<td>utilde</td>
<td>1021</td>
</tr>
<tr>
<td>umacron</td>
<td>1022</td>
</tr>
<tr>
<td>overline</td>
<td>1150</td>
</tr>
<tr>
<td>kana_fullstop</td>
<td>1185</td>
</tr>
<tr>
<td>kana_openingbracket</td>
<td>1186</td>
</tr>
<tr>
<td>kana_closingbracket</td>
<td>1187</td>
</tr>
<tr>
<td>kana_comma</td>
<td>1188</td>
</tr>
<tr>
<td>kana_middledot</td>
<td>1189</td>
</tr>
<tr>
<td>kana_WO</td>
<td>1190</td>
</tr>
<tr>
<td>kana_a</td>
<td>1191</td>
</tr>
<tr>
<td>kana_i</td>
<td>1192</td>
</tr>
<tr>
<td>kana_u</td>
<td>1193</td>
</tr>
<tr>
<td>Arabic Symbol</td>
<td>Code</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>Arabic_semicolon</td>
<td>1467</td>
</tr>
<tr>
<td>Arabic_question_mark</td>
<td>1471</td>
</tr>
<tr>
<td>Arabic_hamza</td>
<td>1473</td>
</tr>
<tr>
<td>Arabic_maddaonalef</td>
<td>1474</td>
</tr>
<tr>
<td>Arabic_hamzaonalef</td>
<td>1475</td>
</tr>
<tr>
<td>Arabic_hamzaonwaw</td>
<td>1476</td>
</tr>
<tr>
<td>Arabic_hamzaunderalef</td>
<td>1477</td>
</tr>
<tr>
<td>Arabic_hamzaonyeh</td>
<td>1478</td>
</tr>
<tr>
<td>Arabic_alef</td>
<td>1479</td>
</tr>
<tr>
<td>Arabic_beh</td>
<td>1480</td>
</tr>
<tr>
<td>Arabic_tehmarbuta</td>
<td>1481</td>
</tr>
<tr>
<td>Arabic_teh</td>
<td>1482</td>
</tr>
<tr>
<td>Arabic_theh</td>
<td>1483</td>
</tr>
<tr>
<td>Arabic_jeem</td>
<td>1484</td>
</tr>
<tr>
<td>Arabic_hah</td>
<td>1485</td>
</tr>
<tr>
<td>Arabic_khah</td>
<td>1486</td>
</tr>
<tr>
<td>Arabic_dal</td>
<td>1487</td>
</tr>
<tr>
<td>Arabic_thal</td>
<td>1488</td>
</tr>
<tr>
<td>Arabic_ra</td>
<td>1489</td>
</tr>
<tr>
<td>Arabic_zain</td>
<td>1490</td>
</tr>
<tr>
<td>Arabic_seen</td>
<td>1491</td>
</tr>
<tr>
<td>Arabic_sheen</td>
<td>1492</td>
</tr>
<tr>
<td>Arabic_sad</td>
<td>1493</td>
</tr>
<tr>
<td>Arabic_dad</td>
<td>1494</td>
</tr>
<tr>
<td>Arabic_tah</td>
<td>1495</td>
</tr>
<tr>
<td>Arabic_zah</td>
<td>1496</td>
</tr>
<tr>
<td>Arabic_ain</td>
<td>1497</td>
</tr>
<tr>
<td>Arabic_ghain</td>
<td>1498</td>
</tr>
<tr>
<td>Arabic_tatweel</td>
<td>1504</td>
</tr>
<tr>
<td>Arabic_feh</td>
<td>1505</td>
</tr>
<tr>
<td>Arabic_qaf</td>
<td>1506</td>
</tr>
<tr>
<td>Arabic_kaf</td>
<td>1507</td>
</tr>
<tr>
<td>Arabic_lam</td>
<td>1508</td>
</tr>
<tr>
<td>Arabic_meem</td>
<td>1509</td>
</tr>
<tr>
<td>Arabic_noon</td>
<td>1510</td>
</tr>
<tr>
<td>Arabic_heh</td>
<td>1511</td>
</tr>
<tr>
<td>Arabic_waw</td>
<td>1512</td>
</tr>
<tr>
<td>Arabic_alefmaksura</td>
<td>1513</td>
</tr>
<tr>
<td>Arabic_yeh</td>
<td>1514</td>
</tr>
<tr>
<td>Arabic_fathatan</td>
<td>1515</td>
</tr>
<tr>
<td>Arabic_dammatan</td>
<td>1516</td>
</tr>
<tr>
<td>Arabic_kasratan</td>
<td>1517</td>
</tr>
<tr>
<td>Arabic_fatha</td>
<td>1518</td>
</tr>
<tr>
<td>Arabic_damma</td>
<td>1519</td>
</tr>
<tr>
<td>Arabic_kasra</td>
<td>1520</td>
</tr>
<tr>
<td>Arabic_shadda</td>
<td>1521</td>
</tr>
<tr>
<td>Arabic_sukun</td>
<td>1522</td>
</tr>
<tr>
<td>Serbian_dje</td>
<td>1597</td>
</tr>
<tr>
<td>Macedonia_gje</td>
<td>1698</td>
</tr>
<tr>
<td>Cyrillic io</td>
<td>1699</td>
</tr>
<tr>
<td>Ukrainian_je</td>
<td>1700</td>
</tr>
<tr>
<td>Macedonia_dse</td>
<td>1701</td>
</tr>
<tr>
<td>Ukrainian_i</td>
<td>1702</td>
</tr>
<tr>
<td>Ukrainian_yi</td>
<td>1703</td>
</tr>
</tbody>
</table>
Serbian_je 1704 0x06a8
Serbian_lje 1705 0x06a9
Serbian_nje 1706 0x06aa
Serbian_tshe 1707 0x06ab
Macedonia_kje 1708 0x06ac
Byelorussian_shortu 1710 0x06ae
Serbian_dze 1711 0x06af
numerosign 1712 0x06b0
Serbian_DJE 1713 0x06b1
Macedonia_GJE 1714 0x06b2
Cyrillic_IO 1715 0x06b3
Ukrainian_JE 1716 0x06b4
Macedonia_DSE 1717 0x06b5
Ukrainian_I 1718 0x06b6
Ukrainian_YI 1719 0x06b7
Serbian_JE 1720 0x06b8
Serbian_LJE 1721 0x06b9
Serbian_NJE 1722 0x06ba
Serbian_TSHE 1723 0x06bb
Macedonia_KJE 1724 0x06bc
Byelorussian_SHORTU 1726 0x06be
Serbian_DZE 1727 0x06bf
Cyrillic_yu 1728 0x06c0
Cyrillic_a 1729 0x06c1
Cyrillic_be 1730 0x06c2
Cyrillic_tse 1731 0x06c3
Cyrillic_de 1732 0x06c4
Cyrillic_ie 1733 0x06c5
Cyrillic_ef 1734 0x06c6
Cyrillic_ghe 1735 0x06c7
Cyrillic_ha 1736 0x06c8
Cyrillic_i 1737 0x06c9
Cyrillic_shorti 1738 0x06ca
Cyrillic_ka 1739 0x06cb
Cyrillic_el 1740 0x06cc
Cyrillic_em 1741 0x06cd
Cyrillic_en 1742 0x06ce
Cyrillic_o 1743 0x06cf
Cyrillic_pe 1744 0x06d0
Cyrillic_ya 1745 0x06d1
Cyrillic_er 1746 0x06d2
Cyrillic_es 1747 0x06d3
Cyrillic_te 1748 0x06d4
Cyrillic_u 1749 0x06d5
Cyrillic_zhe 1750 0x06d6
Cyrillic_ve 1751 0x06d7
Cyrillic_softsign 1752 0x06d8
Cyrillic_yeru 1753 0x06d9
Cyrillic_ze 1754 0x06da
Cyrillic_sha 1755 0x06db
Cyrillic_e 1756 0x06dc
Cyrillic_shcha 1757 0x06dd
Cyrillic_che 1758 0x06de
Cyrillic_hardsign 1759 0x06df
Cyrillic_YU 1760 0x06e0
Cyrillic_A 1761   0x06e1
Cyrillic_BE 1762   0x06e2
Cyrillic_TSE 1763  0x06e3
Cyrillic_DE 1764   0x06e4
Cyrillic_IE 1765   0x06e5
Cyrillic_EF 1766   0x06e6
Cyrillic_GHE 1767  0x06e7
Cyrillic_HA 1768   0x06e8
Cyrillic_I 1769   0x06e9
Cyrillic_SHORTI 1770 0x06ea
Cyrillic_KA 1771   0x06eb
Cyrillic_EL 1772   0x06ec
Cyrillic_EM 1773   0x06ed
Cyrillic_EN 1774   0x06ee
Cyrillic_O 1775   0x06ef
Cyrillic_PE 1776   0x06f0
Cyrillic_YA 1777   0x06f1
Cyrillic_ER 1778   0x06f2
Cyrillic_ES 1779   0x06f3
Cyrillic_TE 1780   0x06f4
Cyrillic_U 1781   0x06f5
Cyrillic_ZHE 1782  0x06f6
Cyrillic_VE 1783   0x06f7
Cyrillic_SOFTSIGN 1784 0x06f8
Cyrillic_YERU 1785  0x06f9
Cyrillic_ZE 1786   0x06fa
Cyrillic_SHA 1787   0x06fb
Cyrillic_E 1788   0x06fc
Cyrillic_SHCHA 1789 0x06fd
Cyrillic_CHE 1790  0x06fe
Cyrillic_HARDSIGN 1791 0x06ff
Greek_ALPHAaccent 1953 0x07a1
Greek_EPSILONaccent 1954 0x07a2
Greek_IOTAaccent 1955 0x07a3
Greek_IOTAaccentdiaeresis 1956 0x07a4
Greek_IOTAdiaeresis 1957 0x07a5
Greek_IOTAaccentdiaeresis 1958 0x07a6
Greek_OMICRONaccent 1959 0x07a7
Greek_UPSILONaccent 1960 0x07a8
Greek_UPSILONdieresis 1961 0x07a9
Greek_UPSILONaccentdieresis 1962 0x07aa
Greek_OMEGAaccent 1963 0x07ab
Greek_alphaaccent 1969 0x07b1
Greek_omicronaccent 1970 0x07b2
Greek_etaaccent 1971 0x07b3
Greek_iotaaccent 1972 0x07b4
Greek_iotadieresis 1973 0x07b5
Greek_iotaaccentdieresis 1974 0x07b6
Greek_omicronaccent 1975 0x07b7
Greek_upsilonaccent 1976 0x07b8
Greek_upsilonaccentdieresis 1977 0x07b9
Greek_upsilonaccentdieresis 1978 0x07ba
Greek_omegaaccent 1979 0x07bb
Greek_ALPHA 1985 0x07c1
Greek_BETA 1986 0x07c2

keysyms
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Year</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greek_GAMMA</td>
<td>1987</td>
<td>0x07c3</td>
</tr>
<tr>
<td>Greek_DELTA</td>
<td>1988</td>
<td>0x07c4</td>
</tr>
<tr>
<td>Greek_EPSILON</td>
<td>1989</td>
<td>0x07c5</td>
</tr>
<tr>
<td>Greek_ZETA</td>
<td>1990</td>
<td>0x07c6</td>
</tr>
<tr>
<td>Greek_ETA</td>
<td>1991</td>
<td>0x07c7</td>
</tr>
<tr>
<td>Greek_THETA</td>
<td>1992</td>
<td>0x07c8</td>
</tr>
<tr>
<td>Greek_IOTA</td>
<td>1993</td>
<td>0x07c9</td>
</tr>
<tr>
<td>Greek_KAPPA</td>
<td>1994</td>
<td>0x07ca</td>
</tr>
<tr>
<td>Greek_LAMBDA</td>
<td>1995</td>
<td>0x07cb</td>
</tr>
<tr>
<td>Greek_MU</td>
<td>1996</td>
<td>0x07cc</td>
</tr>
<tr>
<td>Greek_NU</td>
<td>1997</td>
<td>0x07cd</td>
</tr>
<tr>
<td>Greek_XI</td>
<td>1998</td>
<td>0x07ce</td>
</tr>
<tr>
<td>Greek_OMICRON</td>
<td>1999</td>
<td>0x07cf</td>
</tr>
<tr>
<td>Greek_PI</td>
<td>2000</td>
<td>0x07d0</td>
</tr>
<tr>
<td>Greek_RHO</td>
<td>2001</td>
<td>0x07d1</td>
</tr>
<tr>
<td>Greek_SIGMA</td>
<td>2002</td>
<td>0x07d2</td>
</tr>
<tr>
<td>Greek_TAU</td>
<td>2004</td>
<td>0x07d4</td>
</tr>
<tr>
<td>Greek_UPSILON</td>
<td>2005</td>
<td>0x07d5</td>
</tr>
<tr>
<td>Greek_PHI</td>
<td>2006</td>
<td>0x07d6</td>
</tr>
<tr>
<td>Greek_CHI</td>
<td>2007</td>
<td>0x07d7</td>
</tr>
<tr>
<td>Greek_PSI</td>
<td>2008</td>
<td>0x07d8</td>
</tr>
<tr>
<td>Greek_OMEGA</td>
<td>2009</td>
<td>0x07d9</td>
</tr>
<tr>
<td>Greek_alpha</td>
<td>2017</td>
<td>0x07e1</td>
</tr>
<tr>
<td>Greek_beta</td>
<td>2018</td>
<td>0x07e2</td>
</tr>
<tr>
<td>Greek_gamma</td>
<td>2019</td>
<td>0x07e3</td>
</tr>
<tr>
<td>Greek_delta</td>
<td>2020</td>
<td>0x07e4</td>
</tr>
<tr>
<td>Greek_epsilon</td>
<td>2021</td>
<td>0x07e5</td>
</tr>
<tr>
<td>Greek_zeta</td>
<td>2022</td>
<td>0x07e6</td>
</tr>
<tr>
<td>Greek_eta</td>
<td>2023</td>
<td>0x07e7</td>
</tr>
<tr>
<td>Greek_theta</td>
<td>2024</td>
<td>0x07e8</td>
</tr>
<tr>
<td>Greek_iota</td>
<td>2025</td>
<td>0x07e9</td>
</tr>
<tr>
<td>Greek_kappa</td>
<td>2026</td>
<td>0x07ea</td>
</tr>
<tr>
<td>Greek_lambda</td>
<td>2027</td>
<td>0x07eb</td>
</tr>
<tr>
<td>Greek_mu</td>
<td>2028</td>
<td>0x07ec</td>
</tr>
<tr>
<td>Greek_nu</td>
<td>2029</td>
<td>0x07ed</td>
</tr>
<tr>
<td>Greek_xi</td>
<td>2030</td>
<td>0x07ee</td>
</tr>
<tr>
<td>Greek_omicron</td>
<td>2031</td>
<td>0x07ef</td>
</tr>
<tr>
<td>Greek_pi</td>
<td>2032</td>
<td>0x07f0</td>
</tr>
<tr>
<td>Greek_rho</td>
<td>2033</td>
<td>0x07f1</td>
</tr>
<tr>
<td>Greek_sigma</td>
<td>2034</td>
<td>0x07f2</td>
</tr>
<tr>
<td>Greek_finalsmallsigma</td>
<td>2035</td>
<td>0x07f3</td>
</tr>
<tr>
<td>GreekTau</td>
<td>2036</td>
<td>0x07f4</td>
</tr>
<tr>
<td>Greek_upsilon</td>
<td>2037</td>
<td>0x07f5</td>
</tr>
<tr>
<td>Greek_phi</td>
<td>2038</td>
<td>0x07f6</td>
</tr>
<tr>
<td>Greek_chic</td>
<td>2039</td>
<td>0x07f7</td>
</tr>
<tr>
<td>Greek_psi</td>
<td>2040</td>
<td>0x07f8</td>
</tr>
<tr>
<td>Greek_omega</td>
<td>2041</td>
<td>0x07f9</td>
</tr>
<tr>
<td>leftradicical</td>
<td>2209</td>
<td>0x08a1</td>
</tr>
<tr>
<td>topleftradicical</td>
<td>2210</td>
<td>0x08a2</td>
</tr>
<tr>
<td>horizconnector</td>
<td>2211</td>
<td>0x08a3</td>
</tr>
<tr>
<td>topintegral</td>
<td>2212</td>
<td>0x08a4</td>
</tr>
<tr>
<td>botintegral</td>
<td>2213</td>
<td>0x08a5</td>
</tr>
<tr>
<td>vertconnector</td>
<td>2214</td>
<td>0x08a6</td>
</tr>
<tr>
<td>topleftsqbracket</td>
<td>2215</td>
<td>0x08a7</td>
</tr>
<tr>
<td>botleftsqbracket</td>
<td>2216</td>
<td>0x08a8</td>
</tr>
<tr>
<td>Key Sequence</td>
<td>Code Point</td>
<td></td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------------</td>
<td></td>
</tr>
<tr>
<td><code>toprightsqbracket</code></td>
<td>2217</td>
<td></td>
</tr>
<tr>
<td><code>botrightsqbracket</code></td>
<td>2218</td>
<td></td>
</tr>
<tr>
<td><code>topleftparens</code></td>
<td>2219</td>
<td></td>
</tr>
<tr>
<td><code>botleftparens</code></td>
<td>2220</td>
<td></td>
</tr>
<tr>
<td><code>toprightparens</code></td>
<td>2221</td>
<td></td>
</tr>
<tr>
<td><code>botrightparens</code></td>
<td>2222</td>
<td></td>
</tr>
<tr>
<td><code>leftmiddlecurlybrace</code></td>
<td>2223</td>
<td></td>
</tr>
<tr>
<td><code>rightmiddlecurlybrace</code></td>
<td>2224</td>
<td></td>
</tr>
<tr>
<td><code>topleftsummation</code></td>
<td>2225</td>
<td></td>
</tr>
<tr>
<td><code>botleftsummation</code></td>
<td>2226</td>
<td></td>
</tr>
<tr>
<td><code>toprightsummation</code></td>
<td>2227</td>
<td></td>
</tr>
<tr>
<td><code>botrightsummation</code></td>
<td>2228</td>
<td></td>
</tr>
<tr>
<td><code>topvertsummationconnector</code></td>
<td>2229</td>
<td></td>
</tr>
<tr>
<td><code>botvertsummationconnector</code></td>
<td>2230</td>
<td></td>
</tr>
<tr>
<td><code>lessthanequal</code></td>
<td>2236</td>
<td></td>
</tr>
<tr>
<td><code>notequal</code></td>
<td>2237</td>
<td></td>
</tr>
<tr>
<td><code>greaterthanequal</code></td>
<td>2238</td>
<td></td>
</tr>
<tr>
<td><code>integral</code></td>
<td>2239</td>
<td></td>
</tr>
<tr>
<td><code>therefore</code></td>
<td>2240</td>
<td></td>
</tr>
<tr>
<td><code>variation</code></td>
<td>2241</td>
<td></td>
</tr>
<tr>
<td><code>infinity</code></td>
<td>2242</td>
<td></td>
</tr>
<tr>
<td><code>nabla</code></td>
<td>2245</td>
<td></td>
</tr>
<tr>
<td><code>approximate</code></td>
<td>2248</td>
<td></td>
</tr>
<tr>
<td><code>similarequal</code></td>
<td>2249</td>
<td></td>
</tr>
<tr>
<td><code>ifonlyif</code></td>
<td>2253</td>
<td></td>
</tr>
<tr>
<td><code>implies</code></td>
<td>2254</td>
<td></td>
</tr>
<tr>
<td><code>identical</code></td>
<td>2255</td>
<td></td>
</tr>
<tr>
<td><code>radical</code></td>
<td>2262</td>
<td></td>
</tr>
<tr>
<td><code>includedin</code></td>
<td>2266</td>
<td></td>
</tr>
<tr>
<td><code>includes</code></td>
<td>2267</td>
<td></td>
</tr>
<tr>
<td><code>intersection</code></td>
<td>2268</td>
<td></td>
</tr>
<tr>
<td><code>union</code></td>
<td>2269</td>
<td></td>
</tr>
<tr>
<td><code>logicaland</code></td>
<td>2270</td>
<td></td>
</tr>
<tr>
<td><code>logicalor</code></td>
<td>2271</td>
<td></td>
</tr>
<tr>
<td><code>partialderivative</code></td>
<td>2287</td>
<td></td>
</tr>
<tr>
<td><code>function</code></td>
<td>2294</td>
<td></td>
</tr>
<tr>
<td><code>leftarrow</code></td>
<td>2299</td>
<td></td>
</tr>
<tr>
<td><code>uparrow</code></td>
<td>2300</td>
<td></td>
</tr>
<tr>
<td><code>rightarrow</code></td>
<td>2301</td>
<td></td>
</tr>
<tr>
<td><code>downarrow</code></td>
<td>2302</td>
<td></td>
</tr>
<tr>
<td><code>blank</code></td>
<td>2527</td>
<td></td>
</tr>
<tr>
<td><code>soliddiamond</code></td>
<td>2528</td>
<td></td>
</tr>
<tr>
<td><code>checkerboard</code></td>
<td>2529</td>
<td></td>
</tr>
<tr>
<td><code>ht</code></td>
<td>2530</td>
<td></td>
</tr>
<tr>
<td><code>ff</code></td>
<td>2531</td>
<td></td>
</tr>
<tr>
<td><code>cr</code></td>
<td>2532</td>
<td></td>
</tr>
<tr>
<td><code>lf</code></td>
<td>2533</td>
<td></td>
</tr>
<tr>
<td><code>nl</code></td>
<td>2536</td>
<td></td>
</tr>
<tr>
<td><code>vt</code></td>
<td>2537</td>
<td></td>
</tr>
<tr>
<td><code>lowrightcorner</code></td>
<td>2538</td>
<td></td>
</tr>
<tr>
<td><code>uprightcorner</code></td>
<td>2539</td>
<td></td>
</tr>
<tr>
<td><code>upleftcorner</code></td>
<td>2540</td>
<td></td>
</tr>
<tr>
<td><code>lowleftcorner</code></td>
<td>2541</td>
<td></td>
</tr>
</tbody>
</table>

**Keysyms**
<table>
<thead>
<tr>
<th>Key</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>crossinglines</td>
<td>2542</td>
<td>0x09ee</td>
</tr>
<tr>
<td>horizlinescan1</td>
<td>2543</td>
<td>0x09ef</td>
</tr>
<tr>
<td>horizlinescan3</td>
<td>2544</td>
<td>0x09f0</td>
</tr>
<tr>
<td>horizlinescan5</td>
<td>2545</td>
<td>0x09f1</td>
</tr>
<tr>
<td>horizlinescan7</td>
<td>2546</td>
<td>0x09f2</td>
</tr>
<tr>
<td>horizlinescan9</td>
<td>2547</td>
<td>0x09f3</td>
</tr>
<tr>
<td>leftt</td>
<td>2548</td>
<td>0x09f4</td>
</tr>
<tr>
<td>rightt</td>
<td>2549</td>
<td>0x09f5</td>
</tr>
<tr>
<td>bott</td>
<td>2550</td>
<td>0x09f6</td>
</tr>
<tr>
<td>topt</td>
<td>2551</td>
<td>0x09f7</td>
</tr>
<tr>
<td>vertbar</td>
<td>2552</td>
<td>0x09f8</td>
</tr>
<tr>
<td>emspace</td>
<td>2721</td>
<td>0x0aa1</td>
</tr>
<tr>
<td>enspace</td>
<td>2722</td>
<td>0x0aa2</td>
</tr>
<tr>
<td>em3space</td>
<td>2723</td>
<td>0x0aa3</td>
</tr>
<tr>
<td>em4space</td>
<td>2724</td>
<td>0x0aa4</td>
</tr>
<tr>
<td>digitspace</td>
<td>2725</td>
<td>0x0aa5</td>
</tr>
<tr>
<td>punctspace</td>
<td>2726</td>
<td>0x0aa6</td>
</tr>
<tr>
<td>thinspace</td>
<td>2727</td>
<td>0x0aa7</td>
</tr>
<tr>
<td>hairspace</td>
<td>2728</td>
<td>0x0aa8</td>
</tr>
<tr>
<td>emdash</td>
<td>2729</td>
<td>0x0aa9</td>
</tr>
<tr>
<td>endash</td>
<td>2730</td>
<td>0x0aaa</td>
</tr>
<tr>
<td>signifblank</td>
<td>2732</td>
<td>0x0aac</td>
</tr>
<tr>
<td>ellipsis</td>
<td>2734</td>
<td>0x0aae</td>
</tr>
<tr>
<td>doubbaslinelinedot</td>
<td>2735</td>
<td>0x0aaf</td>
</tr>
<tr>
<td>onethird</td>
<td>2736</td>
<td>0x0ab0</td>
</tr>
<tr>
<td>twothirds</td>
<td>2737</td>
<td>0x0ab1</td>
</tr>
<tr>
<td>onefifth</td>
<td>2738</td>
<td>0x0ab2</td>
</tr>
<tr>
<td>twofifths</td>
<td>2739</td>
<td>0x0ab3</td>
</tr>
<tr>
<td>threefifths</td>
<td>2740</td>
<td>0x0ab4</td>
</tr>
<tr>
<td>fourfifths</td>
<td>2741</td>
<td>0x0ab5</td>
</tr>
<tr>
<td>onesixth</td>
<td>2742</td>
<td>0x0ab6</td>
</tr>
<tr>
<td>fivesixths</td>
<td>2743</td>
<td>0x0ab7</td>
</tr>
<tr>
<td>careof</td>
<td>2744</td>
<td>0x0ab8</td>
</tr>
<tr>
<td>figdash</td>
<td>2747</td>
<td>0x0abb</td>
</tr>
<tr>
<td>leftanglebracket</td>
<td>2748</td>
<td>0x0abc</td>
</tr>
<tr>
<td>decimalpoint</td>
<td>2749</td>
<td>0x0abd</td>
</tr>
<tr>
<td>rightanglebracket</td>
<td>2750</td>
<td>0x0abe</td>
</tr>
<tr>
<td>marker</td>
<td>2751</td>
<td>0x0abf</td>
</tr>
<tr>
<td>oneeighth</td>
<td>2755</td>
<td>0x0ac3</td>
</tr>
<tr>
<td>threeeighths</td>
<td>2756</td>
<td>0x0ac4</td>
</tr>
<tr>
<td>fiveeighths</td>
<td>2757</td>
<td>0x0ac5</td>
</tr>
<tr>
<td>seveneighths</td>
<td>2758</td>
<td>0x0ac6</td>
</tr>
<tr>
<td>trademark</td>
<td>2761</td>
<td>0x0ac9</td>
</tr>
<tr>
<td>signaturemark</td>
<td>2762</td>
<td>0x0aca</td>
</tr>
<tr>
<td>trademarkincircle</td>
<td>2763</td>
<td>0x0acb</td>
</tr>
<tr>
<td>leftopentriangle</td>
<td>2764</td>
<td>0x0acc</td>
</tr>
<tr>
<td>rightopentriangle</td>
<td>2765</td>
<td>0x0acd</td>
</tr>
<tr>
<td>emopencircle</td>
<td>2766</td>
<td>0x0ace</td>
</tr>
<tr>
<td>emopenrectangle</td>
<td>2767</td>
<td>0x0acf</td>
</tr>
<tr>
<td>leftsinglequotemark</td>
<td>2768</td>
<td>0x0ad0</td>
</tr>
<tr>
<td>rightsinglequotemark</td>
<td>2769</td>
<td>0x0ad1</td>
</tr>
<tr>
<td>leftdoublequotemark</td>
<td>2770</td>
<td>0x0ad2</td>
</tr>
<tr>
<td>rightdoublequotemark</td>
<td>2771</td>
<td>0x0ad3</td>
</tr>
<tr>
<td>prescription</td>
<td>2772</td>
<td>0x0ad4</td>
</tr>
<tr>
<td>minutes</td>
<td>2774</td>
<td>0x0ad6</td>
</tr>
<tr>
<td>Keysym</td>
<td>Code</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
<td></td>
</tr>
<tr>
<td>seconds</td>
<td>2775</td>
<td></td>
</tr>
<tr>
<td>latin cross</td>
<td>2777</td>
<td></td>
</tr>
<tr>
<td>hexagram</td>
<td>2778</td>
<td></td>
</tr>
<tr>
<td>filledrectbullet</td>
<td>2779</td>
<td></td>
</tr>
<tr>
<td>filledlefttribullet</td>
<td>2780</td>
<td></td>
</tr>
<tr>
<td>filledrighttribullet</td>
<td>2781</td>
<td></td>
</tr>
<tr>
<td>emfilledcircle</td>
<td>2782</td>
<td></td>
</tr>
<tr>
<td>emfilledrect</td>
<td>2783</td>
<td></td>
</tr>
<tr>
<td>enopencircbullet</td>
<td>2784</td>
<td></td>
</tr>
<tr>
<td>enopensquarebullet</td>
<td>2785</td>
<td></td>
</tr>
<tr>
<td>openrectbullet</td>
<td>2786</td>
<td></td>
</tr>
<tr>
<td>opentribullet</td>
<td>2787</td>
<td></td>
</tr>
<tr>
<td>opentribulletdown</td>
<td>2788</td>
<td></td>
</tr>
<tr>
<td>openstar</td>
<td>2789</td>
<td></td>
</tr>
<tr>
<td>enfilledcircbullet</td>
<td>2790</td>
<td></td>
</tr>
<tr>
<td>enfilledsqbullet</td>
<td>2791</td>
<td></td>
</tr>
<tr>
<td>filledtribulletup</td>
<td>2792</td>
<td></td>
</tr>
<tr>
<td>filledtribulletdown</td>
<td>2793</td>
<td></td>
</tr>
<tr>
<td>leftpointer</td>
<td>2794</td>
<td></td>
</tr>
<tr>
<td>rightpointer</td>
<td>2795</td>
<td></td>
</tr>
<tr>
<td>club</td>
<td>2796</td>
<td></td>
</tr>
<tr>
<td>diamond</td>
<td>2797</td>
<td></td>
</tr>
<tr>
<td>heart</td>
<td>2798</td>
<td></td>
</tr>
<tr>
<td>maltesecross</td>
<td>2800</td>
<td></td>
</tr>
<tr>
<td>dagger</td>
<td>2801</td>
<td></td>
</tr>
<tr>
<td>doubledagger</td>
<td>2802</td>
<td></td>
</tr>
<tr>
<td>checkmark</td>
<td>2803</td>
<td></td>
</tr>
<tr>
<td>ballotcross</td>
<td>2804</td>
<td></td>
</tr>
<tr>
<td>musicalsharp</td>
<td>2805</td>
<td></td>
</tr>
<tr>
<td>musicalflat</td>
<td>2806</td>
<td></td>
</tr>
<tr>
<td>malesymbol</td>
<td>2807</td>
<td></td>
</tr>
<tr>
<td>femalesymbol</td>
<td>2808</td>
<td></td>
</tr>
<tr>
<td>telephone</td>
<td>2809</td>
<td></td>
</tr>
<tr>
<td>phonographcopyright</td>
<td>2810</td>
<td></td>
</tr>
<tr>
<td>caret</td>
<td>2811</td>
<td></td>
</tr>
<tr>
<td>singlelowquotemark</td>
<td>2812</td>
<td></td>
</tr>
<tr>
<td>doublelowquotemark</td>
<td>2813</td>
<td></td>
</tr>
<tr>
<td>cursor</td>
<td>2814</td>
<td></td>
</tr>
<tr>
<td>leftcaret</td>
<td>2815</td>
<td></td>
</tr>
<tr>
<td>rightcaret</td>
<td>2816</td>
<td></td>
</tr>
<tr>
<td>downcaret</td>
<td>2817</td>
<td></td>
</tr>
<tr>
<td>upcaret</td>
<td>2818</td>
<td></td>
</tr>
<tr>
<td>overbar</td>
<td>2819</td>
<td></td>
</tr>
<tr>
<td>downtack</td>
<td>2820</td>
<td></td>
</tr>
<tr>
<td>upshoe</td>
<td>2821</td>
<td></td>
</tr>
<tr>
<td>downstile</td>
<td>2822</td>
<td></td>
</tr>
<tr>
<td>underbar</td>
<td>2823</td>
<td></td>
</tr>
<tr>
<td>jot</td>
<td>2824</td>
<td></td>
</tr>
<tr>
<td>quad</td>
<td>2825</td>
<td></td>
</tr>
<tr>
<td>uptack</td>
<td>2826</td>
<td></td>
</tr>
<tr>
<td>circle</td>
<td>2827</td>
<td></td>
</tr>
<tr>
<td>upstile</td>
<td>2828</td>
<td></td>
</tr>
<tr>
<td>downshoe</td>
<td>2829</td>
<td></td>
</tr>
<tr>
<td>rightshoe</td>
<td>2830</td>
<td></td>
</tr>
</tbody>
</table>
leftshoe                          3034     0x0bda
lefttack                          3036     0x0bdc
righttack                         3068     0x0bfc
hebrew_aleph                      3296     0xce0
hebrew_beth                        3297     0xce1
hebrew_gimmel                     3298     0xce2
hebrew_daleth                     3299     0xce3
hebrew_he                          3300     0xce4
hebrew_waw                        3301     0xce5
hebrew_zayin                      3302     0xce6
hebrew_het                         3303     0xce7
hebrew_teth                        3304     0xce8
hebrew_yod                         3305     0xce9
hebrew_finalkaph                  3306     0xcea
hebrew_kaph                       3307     0ceb
hebrew_lamed                      3308     0xcec
hebrew_finalmem                    3309     0xced
hebrew_mem                         3310     0xcee
hebrew_finalnun                   3311     0cef
hebrew_nun                         3312     0xcf0
hebrew_samekh                     3313     0xcf1
hebrew_ayin                       3314     0xcf2
hebrew_finalpe                    3315     0xcf3
hebrew_pe                          3316     0xcf4
hebrew_finalzadi                   3317     0xcf5
hebrew_zadi                        3318     0xcf6
hebrew_kuf                         3319     0xcf7
hebrew_resh                        3320     0xcf8
hebrew_shin                       3321     0xcf9
hebrew_taf                         3322     0xcfa
BackSpace                        65288     0xff08
Tab                               65289     0xff09
Linefeed                         65290     0xff0a
Clear                            65291     0xff0b
Return                          65293     0xff0d
Pause                            65299     0xff13
Scroll_Lock                       65300     0xff14
Sys_Req                           65301     0xff15
Escape                           65307     0xff1b
Multi_key                       65312     0xff20
Kanji                             65313     0xff21
Home                             65360     0xff50
Left                             65361     0xff51
Up                               65362     0xff52
Right                            65363     0xff53
Down                             65364     0xff54
Prior                            65365     0xff55
Next                             65366     0xff56
End                              65367     0xff57
Begin                            65368     0xff58
Win_L                           65371     0xff5b
Win_R                           65372     0xff5c
App                             65373     0xff5d
Select                          65376     0xff60

keysyms 504
<table>
<thead>
<tr>
<th>Key</th>
<th>Code</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>Print</td>
<td>65377</td>
<td>0xff61</td>
</tr>
<tr>
<td>Execute</td>
<td>65378</td>
<td>0xff62</td>
</tr>
<tr>
<td>Insert</td>
<td>65379</td>
<td>0xff63</td>
</tr>
<tr>
<td>Undo</td>
<td>65381</td>
<td>0xff65</td>
</tr>
<tr>
<td>Redo</td>
<td>65382</td>
<td>0xff66</td>
</tr>
<tr>
<td>Menu</td>
<td>65383</td>
<td>0xff67</td>
</tr>
<tr>
<td>Find</td>
<td>65384</td>
<td>0xff68</td>
</tr>
<tr>
<td>Cancel</td>
<td>65385</td>
<td>0xff69</td>
</tr>
<tr>
<td>Help</td>
<td>65386</td>
<td>0xff6a</td>
</tr>
<tr>
<td>Break</td>
<td>65387</td>
<td>0xff6b</td>
</tr>
<tr>
<td>Hebrew_switch</td>
<td>65406</td>
<td>0xff7e</td>
</tr>
<tr>
<td>Num_Lock</td>
<td>65407</td>
<td>0xff7f</td>
</tr>
<tr>
<td>KP_Space</td>
<td>65408</td>
<td>0xff80</td>
</tr>
<tr>
<td>KP_Tab</td>
<td>65417</td>
<td>0xff89</td>
</tr>
<tr>
<td>KP_Enter</td>
<td>65421</td>
<td>0xff8d</td>
</tr>
<tr>
<td>KP_F1</td>
<td>65425</td>
<td>0xff91</td>
</tr>
<tr>
<td>KP_F2</td>
<td>65426</td>
<td>0xff92</td>
</tr>
<tr>
<td>KP_F3</td>
<td>65427</td>
<td>0xff93</td>
</tr>
<tr>
<td>KP_F4</td>
<td>65428</td>
<td>0xff94</td>
</tr>
<tr>
<td>KP_Multiply</td>
<td>65450</td>
<td>0xff9a</td>
</tr>
<tr>
<td>KP_Add</td>
<td>65451</td>
<td>0xffab</td>
</tr>
<tr>
<td>KP_Separator</td>
<td>65452</td>
<td>0xffac</td>
</tr>
<tr>
<td>KP_Subtract</td>
<td>65453</td>
<td>0xffad</td>
</tr>
<tr>
<td>KP.Decimal</td>
<td>65454</td>
<td>0xffae</td>
</tr>
<tr>
<td>KP_Divide</td>
<td>65455</td>
<td>0xffaf</td>
</tr>
<tr>
<td>KP_0</td>
<td>65456</td>
<td>0xffb0</td>
</tr>
<tr>
<td>KP_1</td>
<td>65457</td>
<td>0xffb1</td>
</tr>
<tr>
<td>KP_2</td>
<td>65458</td>
<td>0xffb2</td>
</tr>
<tr>
<td>KP_3</td>
<td>65459</td>
<td>0xffb3</td>
</tr>
<tr>
<td>KP_4</td>
<td>65460</td>
<td>0xffb4</td>
</tr>
<tr>
<td>KP_5</td>
<td>65461</td>
<td>0xffb5</td>
</tr>
<tr>
<td>KP_6</td>
<td>65462</td>
<td>0xffb6</td>
</tr>
<tr>
<td>KP_7</td>
<td>65463</td>
<td>0xffb7</td>
</tr>
<tr>
<td>KP_8</td>
<td>65464</td>
<td>0xffb8</td>
</tr>
<tr>
<td>KP_9</td>
<td>65465</td>
<td>0xffb9</td>
</tr>
<tr>
<td>KP_Equal</td>
<td>65469</td>
<td>0xffbd</td>
</tr>
<tr>
<td>F1</td>
<td>65470</td>
<td>0xffbe</td>
</tr>
<tr>
<td>F2</td>
<td>65471</td>
<td>0xffbf</td>
</tr>
<tr>
<td>F3</td>
<td>65472</td>
<td>0xffc0</td>
</tr>
<tr>
<td>F4</td>
<td>65473</td>
<td>0xffc1</td>
</tr>
<tr>
<td>F5</td>
<td>65474</td>
<td>0xffc2</td>
</tr>
<tr>
<td>F6</td>
<td>65475</td>
<td>0xffc3</td>
</tr>
<tr>
<td>F7</td>
<td>65476</td>
<td>0xffc4</td>
</tr>
<tr>
<td>F8</td>
<td>65477</td>
<td>0xffc5</td>
</tr>
<tr>
<td>F9</td>
<td>65478</td>
<td>0xffc6</td>
</tr>
<tr>
<td>F10</td>
<td>65479</td>
<td>0xffc7</td>
</tr>
<tr>
<td>L1</td>
<td>65480</td>
<td>0xffc8</td>
</tr>
<tr>
<td>L2</td>
<td>65481</td>
<td>0xffc9</td>
</tr>
<tr>
<td>L3</td>
<td>65482</td>
<td>0xffca</td>
</tr>
<tr>
<td>L4</td>
<td>65483</td>
<td>0xffcb</td>
</tr>
<tr>
<td>L5</td>
<td>65484</td>
<td>0xffcc</td>
</tr>
<tr>
<td>L6</td>
<td>65485</td>
<td>0xffcd</td>
</tr>
<tr>
<td>L7</td>
<td>65486</td>
<td>0xffce</td>
</tr>
<tr>
<td>L8</td>
<td>65487</td>
<td>0xffcf</td>
</tr>
<tr>
<td>L9</td>
<td>65488</td>
<td>0xffd0</td>
</tr>
</tbody>
</table>
L10 65489 0xffd1
R1 65490 0xffd2
R2 65491 0xffd3
R3 65492 0xffd4
R4 65493 0xffd5
R5 65494 0xffd6
R6 65495 0xffd7
R7 65496 0xffd8
R8 65497 0xffd9
R9 65498 0xffda
R10 65499 0xffdb
R11 65500 0xffdc
R12 65501 0xffdd
F33 65502 0xffde
R14 65503 0xffdf
R15 65504 0xffe0
Shift_L 65505 0xffe1
Shift_R 65506 0xffe2
Control_L 65507 0xffe3
Control_R 65508 0xffe4
Caps_Lock 65509 0xffe5
Shift_Lock 65510 0xffe6
Meta_L 65511 0xffe7
Meta_R 65512 0xffe8
Alt_L 65513 0xffe9
Alt_R 65514 0xffea
Super_L 65515 0xffeb
Super_R 65516 0xffec
Hyper_L 65517 0xffed
Hyper_R 65518 0xffee
Delete 65535 0xffff

SEE ALSO

bind

KEYWORDS

keysym, bind, binding

Copyright ' 1995-1997 Roger E. Critchlow Jr.
**NAME**

label – Create and manipulate label widgets

**SYNOPSIS**

```
label
```

**STANDARD OPTIONS**

- `activebackground`, `activeBackground`, `Foreground`
- `activeforeground`, `activeForeground`, `Background`
- `anchor`, `anchor`, `Anchor`
- `background` or `-bg`, `background`, `Background`
- `bitmap`, `bitmap`, `Bitmap`
- `borderwidth` or `-bd`, `borderWidth`, `BorderWidth`
- `compound`, `compound`, `Compound`
- `cursor`, `cursor`, `Cursor`
- `disabledforeground`, `disabledForeground`, `DisabledForeground`
- `font`, `font`, `Font`
- `foreground` or `-fg`, `foreground`, `Foreground`
- `highlightbackground`, `highlightBackground`, `HighlightBackground`
- `highlightcolor`, `highlightColor`, `HighlightColor`
- `highlightthickness`, `highlightThickness`, `HighlightThickness`
- `image`, `image`, `Image`
- `justify`, `justify`, `Justify`
- `padx`, `padX`, `Pad`
- `pady`, `padY`, `Pad`
- `relief`, `relief`, `Relief`
- `takefocus`, `takeFocus`, `TakeFocus`
- `text`, `text`, `Text`
- `textvariable`, `textVariable`, `Variable`
- `underline`, `underline`, `Underline`
- `wraplength`, `wrapLength`, `WrapLength`

**WIDGET–SPECIFIC OPTIONS**

- `height`, `height`, `Height`
- `state`, `state`, `State`
- `width`, `width`, `Width`

**DESCRIPTION**

**WIDGET COMMAND**

```
pathName cget option
```

```
pathName configure ?option? ?value option value ...?
```

**BINDINGS**

**KEYWORDS**
NAME

label – Create and manipulate label widgets

SYNOPSIS

label pathName ?options?

STANDARD OPTIONS

-activebackground, activeBackground, Foreground
-activeforeground, activeForeground, Background
-anchor, anchor, Anchor
-background or –bg, background, Background
-bitmap, bitmap, Bitmap
-borderwidth or –bd, borderWidth, BorderWidth
-compound, compound, Compound
-cursor, cursor, Cursor
-disabledforeground, disabledForeground, DisabledForeground
-font, font, Font
-foreground or –fg, foreground, Foreground
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-image, image, Image
-justify, justify, Justify
-padx, padX, Pad
-pady, padY, Pad
-relief, relief, Relief
-takefocus, takeFocus, TakeFocus
-text, text, Text
-textvariable, textVariable, Variable
-underline, underline, Underline
-wraplength, wrapLength, WrapLength

WIDGET–SPECIFIC OPTIONS

Command–Line Name: –height
Database Name: height
Database Class: Height

Specifies a desired height for the label. If an image or bitmap is being displayed in the label then the value is in screen units (i.e. any of the forms acceptable to Tk_GetPixels); for text it is in lines of text. If this option isn't specified, the label's desired height is computed from the size of the image or bitmap or text being displayed in it.

Command–Line Name: –state
Database Name: state
Database Class: State
  Specifies one of three states for the label: normal, active, or disabled. In normal state the button is displayed using the foreground and background options. In active state the label is displayed using the activeForeground and activeBackground options. In the disabled state the disabledForeground and background options determine how the button is displayed.
Command-Line Name: --width
Database Name: width
Database Class: Width
  Specifies a desired width for the label. If an image or bitmap is being displayed in the label then the value is in screen units (i.e. any of the forms acceptable to Tk::GetPixels); for text it is in characters. If this option isn't specified, the label's desired width is computed from the size of the image or bitmap or text being displayed in it.

DESCRIPTION

The label command creates a new window (given by the pathName argument) and makes it into a label widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the label such as its colors, font, text, and initial relief. The label command returns its pathName argument. At the time this command is invoked, there must not exist a window named pathName, but pathName's parent must exist.

A label is a widget that displays a textual string, bitmap or image. If text is displayed, it must all be in a single font, but it can occupy multiple lines on the screen (if it contains newlines or if wrapping occurs because of the wrapLength option) and one of the characters may optionally be underlined using the underline option. The label can be manipulated in a few simple ways, such as changing its relief or text, using the commands described below.

WIDGET COMMAND

The label command creates a new Tcl command whose name is pathName. This command may be used to invoke various operations on the widget. It has the following general form:

pathName option ?arg arg ...?

Option and the args determine the exact behavior of the command. The following commands are possible for label widgets:

pathName cget option
  Returns the current value of the configuration option given by option. Option may have any of the values accepted by the label command.
pathName configure ?option? ?value option value ...?
  Query or modify the configuration options of the widget. If no option is specified, returns a list describing all of the available options for pathName (see Tk::ConfigureInfo for information on the format of this list). If option is specified with no value, then the command returns a list describing the
one named option (this list will be identical to the corresponding sublist of the value returned if no option is specified). If one or more option–value pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. Option may have any of the values accepted by the label command.

BINDINGS

When a new label is created, it has no default event bindings: labels are not intended to be interactive.

KEYWORDS

label, widget
NAME

labelframe – Create and manipulate labelframe widgets

SYNOPSIS

labelframe pathName ?options?

STANDARD OPTIONS

−borderwidth or −bd, borderWidth, BorderWidth
−cursor, cursor, Cursor
−font, font, Font
−foreground or −fg, foreground, Foreground
−highlightbackground, highlightBackground, HighlightBackground
−highlightcolor, highlightColor, HighlightColor
−highlightthickness, highlightThickness, HighlightThickness
−padx, padX, Pad
−pady, padY, Pad
−relief, relief, Relief
−takefocus, takeFocus, TakeFocus
−text, text, Text

WIDGET−SPECIFIC OPTIONS

−background, background, Background
−class, class, Class
−colormap, colormap, Colormap
−container, container, Container
−height, height, Height
−labelanchor, labelAnchor, LabelAnchor
−labelwidget, labelWidget, LabelWidget
−visual, visual, Visual
−width, width, Width

DESCRIPTION

WIDGET COMMAND

pathName cget option

pathName configure ?option? ?value option value ...?

BINDINGS

SEE ALSO

KEYWORDS

NAME

labelframe – Create and manipulate labelframe widgets
SYNOPSIS

labelframe pathName ?options?

STANDARD OPTIONS

−borderwidth or −bd, borderWidth, BorderWidth
−cursor, cursor, Cursor
−font, font, Font
−foreground or −fg, foreground, Foreground
−highlightbackground, highlightBackground, HighlightBackground
−highlightcolor, highlightColor, HighlightColor
−highlightthickness, highlightThickness, HighlightThickness
−padx, padX, Pad
−pady, padY, Pad
−relief, relief, Relief
−takefocus, takeFocus, TakeFocus
−text, text, Text

WIDGET−SPECIFIC OPTIONS

Command−Line Name: −background
Database Name: background
Database Class: Background

This option is the same as the standard background option except that its value may also be specified as an empty string. In this case, the widget will display no background or border, and no colors will be consumed from its colormap for its background and border.

Command−Line Name: −class
Database Name: class
Database Class: Class

Specifies a class for the window. This class will be used when querying the option database for the window's other options, and it will also be used later for other purposes such as bindings. The class option may not be changed with the configure widget command.

Command−Line Name: −colormap
Database Name: colormap
Database Class: Colormap

Specifies a colormap to use for the window. The value may be either new, in which case a new colormap is created for the window and its children, or the name of another window (which must be on the same screen and have the same visual as pathName), in which case the new window will use the colormap from the specified window. If the colormap option is not specified, the new window uses the same colormap as its parent. This option may not be changed with the configure widget command.

Command−Line Name: −container
Database Name: container
Database Class: Container
The value must be a boolean. If true, it means that this window will be used as a container in which some other application will be embedded (for example, a Tk toplevel can be embedded using the −use option). The window will support the appropriate window manager protocols for things like geometry requests. The window should not have any children of its own in this application. This option may not be changed with the configure widget command.

**Command−Line Name:** −height
**Database Name:** height
**Database Class:** Height

Specifies the desired height for the window in any of the forms acceptable to Tk_GetPixels. If this option is less than or equal to zero then the window will not request any size at all.

**Command−Line Name:** −labelanchor
**Database Name:** labelAnchor
**Database Class:** LabelAnchor

Specifies where to place the label. A label is only displayed if the −text option is not the empty string. Valid values for this option are (listing them clockwise) nw, n, ne, en, e, es, se, s,sw, ws, w and wn. The default value is nw.

**Command−Line Name:** −labelwidget
**Database Name:** labelWidget
**Database Class:** LabelWidget

Specifies a widget to use as label. This overrides any −text option. The widget must exist before being used as −labelwidget and if it is not a descendant of this window, it will be raised above it in the stacking order.

**Command−Line Name:** −visual
**Database Name:** visual
**Database Class:** Visual

Specifies visual information for the new window in any of the forms accepted by Tk_GetVisual. If this option is not specified, the new window will use the same visual as its parent. The visual option may not be modified with the configure widget command.

**Command−Line Name:** −width
**Database Name:** width
**Database Class:** Width

Specifies the desired width for the window in any of the forms acceptable to Tk_GetPixels. If this option is less than or equal to zero then the window will not request any size at all.

**DESCRIPTION**

The labelframe command creates a new window (given by the pathName argument) and makes it into a labelframe widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the labelframe such as its background color and relief. The labelframe command returns the path name of the new window.

A labelframe is a simple widget. Its primary purpose is to act as a spacer or container for complex window layouts. It has the features of a frame plus the ability to display a label.
The `labelframe` command creates a new Tcl command whose name is the same as the path name of the labelframe's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

`PathName` is the name of the command, which is the same as the labelframe widget's path name. `Option` and the `args` determine the exact behavior of the command. The following commands are possible for frame widgets:

```
pathName cget option
```

Returns the current value of the configuration option given by `option`. `Option` may have any of the values accepted by the `labelframe` command.

```
pathName configure ?option? ?value option value ...?
```

Query or modify the configuration options of the widget. If no `option` is specified, returns a list describing all of the available options for `pathName` (see `Tk_ConfigureInfo` for information on the format of this list). If `option` is specified with no `value`, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no `option` is specified). If one or more `option−value` pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. `Option` may have any of the values accepted by the `labelframe` command.

**BINDINGS**

When a new labelframe is created, it has no default event bindings: labelframes are not intended to be interactive.

**SEE ALSO**

`frame`, `label`

**KEYWORDS**

`labelframe`, `widget`
listbox

NAME

listbox – Create and manipulate listbox widgets

SYNOPSIS

listbox pathName ?options?

STANDARD OPTIONS

-activestyle
-background or -bg, background, Background
-borderwidth or -bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-disabledforeground, disabledForeground, DisabledForeground
-exportselection, exportSelection, ExportSelection
-font, font, Font
-foreground or -fg, foreground, Foreground
-height
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-relief, relief, Relief
-selectbackground, selectBackground, Foreground
-selectborderwidth, selectBorderWidth, BorderWidth
-selectforeground, selectForeground, Background
-setgrid, setGrid, SetGrid
-state
-takefocus, takeFocus, TakeFocus
-width
-xscrollcommand, xScrollCommand, ScrollCommand
-yscrollcommand, yScrollCommand, ScrollCommand

WIDGET-SPECIFIC OPTIONS

-activestyle, activeStyle, ActiveStyle
-height, height, Height
-listvariable, listVariable, Variable
-selectmode, selectMode, SelectMode
-state, state, State
-width, width, Width

DESCRIPTION

INDICES

number
active
anchor
end
@x,y
WIDGET COMMAND

```
pathName activate index
pathName bbox index
pathName cget option
pathName configure ?option? ?value option value ...?
pathName curselection
pathName delete first ?last?
pathName get first ?last?
pathName index index
pathName insert index ?element element ...?
pathName itemcget index option
pathName itemconfigure index ?option? ?value? ?option value ...?

−background color
−foreground color
−selectbackground color
−selectforeground color
```

```
pathName nearest y
pathName scan option args
pathName scan mark x y
pathName scan dragto x y.
```

```
pathName see index
pathName selection option arg
pathName selection anchor index
pathName selection clear first ?last?
pathName selection includes index
pathName selection set first ?last?
```

```
pathName size
pathName xview args
    pathName xview
    pathName xview index
    pathName xview moveto fraction
    pathName xview scroll number what
```

```
pathName yview ?args?
    pathName yview
    pathName yview index
    pathName yview moveto fraction
    pathName yview scroll number what
```

DEFAULT BINDINGS

KEYWORDS
SYNOPSIS

listbox pathName ?options?

STANDARD OPTIONS

-activestyle
-background or -bg, background, Background
-borderwidth or -bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-disabledforeground, disabledForeground, DisabledForeground
-exportselection, exportSelection, ExportSelection
-font, font, Font
-foreground or -fg, foreground, Foreground
-height
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-relief, relief, Relief
-selectbackground, selectBackground, Foreground
-selectborderwidth, selectBorderWidth, BorderWidth
-selectforeground, selectForeground, Background
-setgrid, setGrid, SetGrid
-state
-takefocus, takeFocus, TakeFocus
-width
-xscrollcommand, xScrollCommand, ScrollCommand
-yscrollcommand, yScrollCommand, ScrollCommand

WIDGET–SPECIFIC OPTIONS

Command–Line Name: -activestyle
Database Name: activeStyle
Database Class: ActiveStyle
Specifies the style in which to draw the active element. This must be one of dotbox (show a focus ring around the active element), none (no special indication of active element) or underline (underline the active element). The default is underline.

Command–Line Name: -height
Database Name: height
Database Class: Height
Specifies the desired height for the window, in lines. If zero or less, then the desired height for the window is made just large enough to hold all the elements in the listbox.

Command–Line Name: -listvariable
Database Name: listVariable
Database Class: Variable

SYNOPSIS
Specifies the name of a variable. The value of the variable is a list to be displayed inside the widget; if the variable value changes then the widget will automatically update itself to reflect the new value. Attempts to assign a variable with an invalid list value to \texttt{listvariable} will cause an error. Attempts to unset a variable in use as a \texttt{listvariable} will fail but will not generate an error.

\textbf{Command–Line Name: } \texttt{selectmode}  \\
\textbf{Database Name: } selectMode  \\
\textbf{Database Class: } SelectMode  \\
Specifies one of several styles for manipulating the selection. The value of the option may be arbitrary, but the default bindings expect it to be either \texttt{single}, \texttt{browse}, \texttt{multiple}, or \texttt{extended}; the default value is \texttt{browse}.

\textbf{Command–Line Name: } \texttt{state}  \\
\textbf{Database Name: } state  \\
\textbf{Database Class: } State  \\
Specifies one of two states for the listbox: \texttt{normal} or \texttt{disabled}. If the listbox is disabled then items may not be inserted or deleted, items are drawn in the \texttt{disabledforeground} color, and selection cannot be modified and is not shown (though selection information is retained).

\textbf{Command–Line Name: } \texttt{width}  \\
\textbf{Database Name: } width  \\
\textbf{Database Class: } Width  \\
Specifies the desired width for the window in characters. If the font doesn't have a uniform width then the width of the character `0' is used in translating from character units to screen units. If zero or less, then the desired width for the window is made just large enough to hold all the elements in the listbox.

\textbf{DESCRIPTION}

The \texttt{listbox} command creates a new window (given by the \texttt{pathName} argument) and makes it into a listbox widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the listbox such as its colors, font, text, and relief. The \texttt{listbox} command returns its \texttt{pathName} argument. At the time this command is invoked, there must not exist a window named \texttt{pathName}, but \texttt{pathName}'s parent must exist.

A listbox is a widget that displays a list of strings, one per line. When first created, a new listbox has no elements. Elements may be added or deleted using widget commands described below. In addition, one or more elements may be selected as described below. If a listbox is exporting its selection (see \texttt{exportSelection} option), then it will observe the standard X11 protocols for handling the selection. Listbox selections are available as type \texttt{STRING}; the value of the selection will be the text of the selected elements, with newlines separating the elements.

It is not necessary for all the elements to be displayed in the listbox window at once; commands described below may be used to change the view in the window. Listboxes allow scrolling in both directions using the standard \texttt{xScrollCommand} and \texttt{yScrollCommand} options. They also support scanning, as described below.
INDICES

Many of the widget commands for listboxes take one or more indices as arguments. An index specifies a particular element of the listbox, in any of the following ways:

- **number**
  Specifies the element as a numerical index, where 0 corresponds to the first element in the listbox.

- **active**
  Indicates the element that has the location cursor. This element will be displayed as specified by `-activestyle` when the listbox has the keyboard focus, and it is specified with the `activate` widget command.

- **anchor**
  Indicates the anchor point for the selection, which is set with the `selection anchor` widget command.

- **end**
  Indicates the end of the listbox. For most commands this refers to the last element in the listbox, but for a few commands such as `index` and `insert` it refers to the element just after the last one.

- **@x,y**
  Indicates the element that covers the point in the listbox window specified by x and y (in pixel coordinates). If no element covers that point, then the closest element to that point is used.

In the widget command descriptions below, arguments named `index`, `first`, and `last` always contain text indices in one of the above forms.

WIDGET COMMAND

The `listbox` command creates a new Tcl command whose name is `pathName`. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

`Option` and the `args` determine the exact behavior of the command. The following commands are possible for listbox widgets:

- **pathName activate index**
  Sets the active element to the one indicated by `index`. If `index` is outside the range of elements in the listbox then the closest element is activated. The active element is drawn as specified by `-activestyle` when the widget has the input focus, and its index may be retrieved with the `index active`.

- **pathName bbox index**
  Returns a list of four numbers describing the bounding box of the text in the element given by `index`. The first two elements of the list give the x and y coordinates of the upper–left corner of the screen area covered by the text (specified in pixels relative to the widget) and the last two elements give the width and height of the area, in pixels. If no part of the element given by `index` is visible on the screen, or if `index` refers to a non–existent element, then the result is an empty string; if the element is partially visible, the result gives the full area of the element, including any parts that are not visible.

- **pathName cget option**
Returns the current value of the configuration option given by option. Option may have any of the
described of the available options for pathName (see Tk_ConfigureInfo for information on the
format of this list). If option is specified with no value, then the command returns a list describing the
one named option (this list will be identical to the corresponding sublist of the value returned if no
option is specified). If one or more option−value pairs are specified, then the command modifies the
given widget option(s) to have the given value(s); in this case the command returns an empty string.
Option may have any of the values accepted by the listbox command.

pathName configure option? ?value option value ...
 Query or modify the configuration options of the widget. If no option is specified, returns a list
that is specifiend to first, i.e. a single element is deleted.

pathName delete first ?last?
 Deletes one or more elements of the listbox. First and last are indices specifying the first and last
elements in the range to delete. If last isn't specified it defaults to first, inclusive. Both first and last may have any of the
standard forms for indices.

pathName index index
 Returns the integer index value that corresponds to index. If index is end the return value is a count of
the number of elements in the listbox (not the index of the last element).

pathName insert index ?element element ...
 Inserts zero or more new elements in the list just before the element given by index. If index is
specified as end then the new elements are added to the end of the list. Returns an empty string.

pathName itemcget index option
 Returns the current value of the item configuration option given by option. Option may have any of the
values accepted by the listbox itemconfigure command.

pathName itemconfigure index ?option? ?value? ?option value ...
 Query or modify the configuration options of an item in the listbox. If no option is specified, returns a list describing all of the available options for the item (see Tk_ConfigureInfo for information on the
format of this list). If option is specified with no value, then the command returns a list describing the
one named option (this list will be identical to the corresponding sublist of the value returned if no
option is specified). If one or more option−value pairs are specified, then the command modifies the
given widget option(s) to have the given value(s); in this case the command returns an empty string.
The following options are currently supported for items:

- background color
  Color specifies the background color to use when displaying the item. It may have any of the
  forms accepted by Tk_GetColor.

- foreground color
  Color specifies the foreground color to use when displaying the item. It may have any of the
  forms accepted by Tk_GetColor.
−selectbackground color
  color specifies the background color to use when displaying the item while it is selected. It may have any of the forms accepted by Tk_GetColor.

−selectforeground color
  color specifies the foreground color to use when displaying the item while it is selected. It may have any of the forms accepted by Tk_GetColor.

pathName nearest y
  Given a y-coordinate within the listbox window, this command returns the index of the (visible) listbox element nearest to that y-coordinate.

pathName scan option args
  This command is used to implement scanning on listboxes. It has two forms, depending on option:

  pathName scan mark x y
    Records x and y and the current view in the listbox window; used in conjunction with later scan dragto commands. Typically this command is associated with a mouse button press in the widget. It returns an empty string.

  pathName scan dragto x y.
    This command computes the difference between its x and y arguments and the x and y arguments to the last scan mark command for the widget. It then adjusts the view by 10 times the difference in coordinates. This command is typically associated with mouse motion events in the widget, to produce the effect of dragging the list at high speed through the window. The return value is an empty string.

pathName see index
  Adjust the view in the listbox so that the element given by index is visible. If the element is already visible then the command has no effect; if the element is near one edge of the window then the listbox scrolls to bring the element into view at the edge; otherwise the listbox scrolls to center the element.

pathName selection option arg
  This command is used to adjust the selection within a listbox. It has several forms, depending on option:

  pathName selection anchor index
    Sets the selection anchor to the element given by index. If index refers to a non-existent element, then the closest element is used. The selection anchor is the end of the selection that is fixed while dragging out a selection with the mouse. The index anchor may be used to refer to the anchor element.

  pathName selection clear first ?last?
    If any of the elements between first and last (inclusive) are selected, they are deselected. The selection state is not changed for elements outside this range.

  pathName selection includes index
    Returns 1 if the element indicated by index is currently selected, 0 if it isn't.

  pathName selection set first ?last?
    Selects all of the elements in the range between first and last, inclusive, without affecting the selection state of elements outside that range.

pathName size
  Returns a decimal string indicating the total number of elements in the listbox.
pathName xview args
This command is used to query and change the horizontal position of the information in the widget's window. It can take any of the following forms:

pathName xview
Returns a list containing two elements. Each element is a real fraction between 0 and 1; together they describe the horizontal span that is visible in the window. For example, if the first element is .2 and the second element is .6, 20% of the listbox's text is off-screen to the left, the middle 40% is visible in the window, and 40% of the text is off-screen to the right. These are the same values passed to scrollbars via the −xscrollcommand option.

pathName xview index
Adjusts the view in the window so that the character position given by index is displayed at the left edge of the window. Character positions are defined by the width of the character 0.

pathName xview moveto fraction
Adjusts the view in the window so that fraction of the total width of the listbox text is off-screen to the left. fraction must be a fraction between 0 and 1.

pathName xview scroll number what
This command shifts the view in the window left or right according to number and what. Number must be an integer. What must be either units or pages or an abbreviation of one of these. If what is units, the view adjusts left or right by number character units (the width of the 0 character) on the display; if it is pages then the view adjusts by number screenfuls. If number is negative then characters farther to the left become visible; if it is positive then characters farther to the right become visible.

pathName yview ?args?
This command is used to query and change the vertical position of the text in the widget's window. It can take any of the following forms:

pathName yview
Returns a list containing two elements, both of which are real fractions between 0 and 1. The first element gives the position of the listbox element at the top of the window, relative to the listbox as a whole (0.5 means it is halfway through the listbox, for example). The second element gives the position of the listbox element just after the last one in the window, relative to the listbox as a whole. These are the same values passed to scrollbars via the −yscrollcommand option.

pathName yview index
Adjusts the view in the window so that the element given by index is displayed at the top of the window.

pathName yview moveto fraction
Adjusts the view in the window so that the element given by fraction appears at the top of the window. Fraction is a fraction between 0 and 1; 0 indicates the first element in the listbox, 0.33 indicates the element one-third the way through the listbox, and so on.

pathName yview scroll number what
This command adjusts the view in the window up or down according to number and what. Number must be an integer. What must be either units or pages. If what is units, the view adjusts up or down by number lines; if it is pages then the view adjusts by number screenfuls.
If \textit{number} is negative then earlier elements become visible; if it is positive then later elements become visible.

**DEFAULT BINDINGS**

Tk automatically creates class bindings for listboxes that give them Motif–like behavior. Much of the behavior of a listbox is determined by its \texttt{selectMode} option, which selects one of four ways of dealing with the selection.

If the selection mode is \texttt{single} or \texttt{browse}, at most one element can be selected in the listbox at once. In both modes, clicking button 1 on an element selects it and deselects any other selected item. In \texttt{browse} mode it is also possible to drag the selection with button 1.

If the selection mode is \texttt{multiple} or \texttt{extended}, any number of elements may be selected at once, including discontiguous ranges. In \texttt{multiple} mode, clicking button 1 on an element toggles its selection state without affecting any other elements. In \texttt{extended} mode, pressing button 1 on an element selects it, deselects everything else, and sets the anchor to the element under the mouse; dragging the mouse with button 1 down extends the selection to include all the elements between the anchor and the element under the mouse, inclusive.

Most people will probably want to use \texttt{browse} mode for single selections and \texttt{extended} mode for multiple selections; the other modes appear to be useful only in special situations.

Any time the selection changes in the listbox, the virtual event \texttt{<<ListboxSelect>>} will be generated. It is easiest to bind to this event to be made aware of any changes to listbox selection.

In addition to the above behavior, the following additional behavior is defined by the default bindings:

\[1\] In \texttt{extended} mode, the selected range can be adjusted by pressing button 1 with the Shift key down: this modifies the selection to consist of the elements between the anchor and the element under the mouse, inclusive. The un–anchored end of this new selection can also be dragged with the button down.

\[2\] In \texttt{extended} mode, pressing button 1 with the Control key down starts a toggle operation: the anchor is set to the element under the mouse, and its selection state is reversed. The selection state of other elements isn't changed. If the mouse is dragged with button 1 down, then the selection state of all elements between the anchor and the element under the mouse is set to match that of the anchor element; the selection state of all other elements remains what it was before the toggle operation began.

\[3\] If the mouse leaves the listbox window with button 1 down, the window scrolls away from the mouse, making information visible that used to be off–screen on the side of the mouse. The scrolling continues until the mouse re–enters the window, the button is released, or the end of the listbox is reached.
Mouse button 2 may be used for scanning. If it is pressed and dragged over the listbox, the contents of the listbox drag at high speed in the direction the mouse moves.

If the Up or Down key is pressed, the location cursor (active element) moves up or down one element. If the selection mode is browse or extended then the new active element is also selected and all other elements are deselected. In extended mode the new active element becomes the selection anchor.

In extended mode, Shift–Up and Shift–Down move the location cursor (active element) up or down one element and also extend the selection to that element in a fashion similar to dragging with mouse button 1.

The Left and Right keys scroll the listbox view left and right by the width of the character 0. Control–Left and Control–Right scroll the listbox view left and right by the width of the window. Control–Prior and Control–Next also scroll left and right by the width of the window.

The Prior and Next keys scroll the listbox view up and down by one page (the height of the window).

The Home and End keys scroll the listbox horizontally to the left and right edges, respectively.

Control–Home sets the location cursor to the first element in the listbox, selects that element, and deselects everything else in the listbox.

Control–End sets the location cursor to the last element in the listbox, selects that element, and deselects everything else in the listbox.

In extended mode, Control–Shift–Home extends the selection to the first element in the listbox and Control–Shift–End extends the selection to the last element.

In multiple mode, Control–Shift–Home moves the location cursor to the first element in the listbox and Control–Shift–End moves the location cursor to the last element.

The space and Select keys make a selection at the location cursor (active element) just as if mouse button 1 had been pressed over this element.

In extended mode, Control–Shift–space and Shift–Select extend the selection to the active element just as if button 1 had been pressed with the Shift key down.

In extended mode, the Escape key cancels the most recent selection and restores all the elements in the selected range to their previous selection state.

Control–slash selects everything in the widget, except in single and browse modes, in which case it selects the active element and deselects everything else.

Control–backslash deselects everything in the widget, except in browse mode where it has no effect.
The F16 key (labelled Copy on many Sun workstations) or Meta–w copies the selection in the widget to the clipboard, if there is a selection.

The behavior of listboxes can be changed by defining new bindings for individual widgets or by redefining the class bindings.

KEYWORDS

listbox, widget

Copyright ' 1990 The Regents of the University of California.
Copyright ' 1994-1997 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
lower

NAME

lower – Change a window’s position in the stacking order

SYNOPSIS

lower window ?belowThis?

DESCRIPTION

If the belowThis argument is omitted then the command lowers window so that it is below all of its siblings in the stacking order (it will be obscured by any siblings that overlap it and will not obscure any siblings). If belowThis is specified then it must be the path name of a window that is either a sibling of window or the descendant of a sibling of window. In this case the lower command will insert window into the stacking order just below belowThis (or the ancestor of belowThis that is a sibling of window); this could end up either raising or lowering window.

SEE ALSO

raise

KEYWORDS

lower, obscure, stacking order

Copyright ' 1990 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

menu, tk_menuSetFocus – Create and manipulate menu widgets

SYNOPSIS

```
menu pathName \?
```

```
tk_menuSetFocus pathName
```

STANDARD OPTIONS

```
- activebackground, activeBackground, Foreground
- activeborderwidth, activeBorderWidth, BorderWidth
- activeforeground, activeForeground, Background
- background or -bg, background, Background
- borderwidth or -bd, borderWidth, BorderWidth
- cursor, cursor, Cursor
- disabledforeground, disabledForeground, DisabledForeground
- font, font, Font
- foreground or -fg, foreground, Foreground
- relief, relief, Relief
- takefocus, takeFocus, TakeFocus
```

WIDGET–SPECIFIC OPTIONS

```
- postcommand, postCommand, Command
- selectcolor, selectColor, Background
- tearoff, tearOff, TearOff
- tearoffcommand, tearOffCommand, TearOffCommand
- title, title, Title
- type, type, Type
```

INTRODUCTION

COMMAND ENTRIES

SEPARATOR ENTRIES

CHECKBUTTON ENTRIES

RADIOBUTTON ENTRIES

CASCADE ENTRIES

TEAR–OFF ENTRIES

MENUBARS

SPECIAL MENUS IN MENUBARS

CLONES

WIDGET COMMAND

```
number
active
end
last
none
@number
```
pattern
pathName activate index
pathName add type ?option value option value ...?
  -activebackground value
  -activeforeground value
  -accelerator value
  -background value
  -bitmap value
  -columnbreak value
  -command value
  -compound value
  -font value
  -foreground value
  -hidemargin value
  -image value
  -indicatoron value
  -label value
  -menu value
  -offvalue value
  -onvalue value
  -selectcolor value
  -selectimage value
  -state value
  -underline value
  -value value
  -variable value
pathName cget option
pathName clone newPPathname ?cloneType?
pathName configure ?option? ?value option value ...?
pathName delete index1 ?index2?
pathName entrycget index option
pathName entryconfigure index ?options?
pathName index index
pathName insert index type ?option value option value ...?
pathName invoke index
pathName post x y
pathName postcascade index
pathName type index
pathName unpost
pathName yposition index

MENU CONFIGURATIONS
  Pulldown Menus in Menubar
  Pulldown Menus in Menu Buttons
  Popup Menus
  Option Menus
Torn−off Menus

DEFAULT BINDINGS

BUGS

KEYWORDS

NAME

menu, tk_menuSetFocus – Create and manipulate menu widgets

SYNOPSIS

menu pathName ?options?
tk_menuSetFocus pathName

STANDARD OPTIONS

−activebackground, activeBackground, Foreground
−activeborderwidth, activeBorderWidth, BorderWidth
−activeforeground, activeForeground, Background
−background or −bg, background, Background
−borderwidth or −bd, borderWidth, BorderWidth
−cursor, cursor, Cursor
−disabledforeground, disabledForeground, DisabledForeground
−font, font, Font
−foreground or −fg, foreground, Foreground
−relief, relief, Relief
−takefocus, takeFocus, TakeFocus

WIDGET−SPECIFIC OPTIONS

Command−Line Name: −postcommand
Database Name: postCommand
Database Class: Command

If this option is specified then it provides a Tcl command to execute each time the menu is posted. The command is invoked by the post widget command before posting the menu. Note that in Tk 8.0 on Macintosh and Windows, all post−commands in a system of menus are executed before any of those menus are posted. This is due to the limitations in the individual platforms' menu managers.

Command−Line Name: −selectcolor
Database Name: selectColor
Database Class: Background

For menu entries that are check buttons or radio buttons, this option specifies the color to display in the indicator when the check button or radio button is selected.

Command−Line Name: −tearoff
Database Name: tearOff
Database Class: TearOff
This option must have a proper boolean value, which specifies whether or not the menu should include a tear-off entry at the top. If so, it will exist as entry 0 of the menu and the other entries will number starting at 1. The default menu bindings arrange for the menu to be torn off when the tear-off entry is invoked.

Command-Line Name: −tearoffcommand
Database Name: tearOffCommand
Database Class: TearOffCommand

If this option has a non-empty value, then it specifies a Tcl command to invoke whenever the menu is torn off. The actual command will consist of the value of this option, followed by a space, followed by the name of the menu window, followed by a space, followed by the name of the name of the torn off menu window. For example, if the option's is "a b" and menu .x.y is torn off to create a new menu .x.tearoff1, then the command "a b .x.y .x.tearoff1" will be invoked.

Command-Line Name: −title
Database Name: title
Database Class: Title

The string will be used to title the window created when this menu is torn off. If the title is NULL, then the window will have the title of the menubutton or the text of the cascade item from which this menu was invoked.

Command-Line Name: −type
Database Name: type
Database Class: Type

This option can be one of menubar, tearoff, or normal, and is set when the menu is created. While the string returned by the configuration database will change if this option is changed, this does not affect the menu widget's behavior. This is used by the cloning mechanism and is not normally set outside of the Tk library.

INTRODUCTION

The menu command creates a new top-level window (given by the pathName argument) and makes it into a menu widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the menu such as its colors and font. The menu command returns its pathName argument. At the time this command is invoked, there must not exist a window named pathName, but pathName's parent must exist.

A menu is a widget that displays a collection of one-line entries arranged in one or more columns. There exist several different types of entries, each with different properties. Entries of different types may be combined in a single menu. Menu entries are not the same as entry widgets. In fact, menu entries are not even distinct widgets; the entire menu is one widget.

Menu entries are displayed with up to three separate fields. The main field is a label in the form of a text string, a bitmap, or an image, controlled by the −label, −bitmap, and −image options for the entry. If the −accelerator option is specified for an entry then a second textual field is displayed to the right of the label. The accelerator typically describes a keystroke sequence that may be typed in the application to cause the same result as invoking the menu entry. The third field is an indicator. The indicator is present only for checkbutton or radiobutton entries. It indicates whether the entry is selected or not, and is displayed to the left
of the entry's string.

In normal use, an entry becomes active (displays itself differently) whenever the mouse pointer is over the entry. If a mouse button is released over the entry then the entry is invoked. The effect of invocation is different for each type of entry; these effects are described below in the sections on individual entries.

Entries may be disabled, which causes their labels and accelerators to be displayed with dimmer colors. The default menu bindings will not allow a disabled entry to be activated or invoked. Disabled entries may be re-enabled, at which point it becomes possible to activate and invoke them again.

Whenever a menu's active entry is changed, a <<MenuSelect>> virtual event is send to the menu. The active item can then be queried from the menu, and an action can be taken, such as setting context-sensitive help text for the entry.

**COMMAND ENTRIES**

The most common kind of menu entry is a command entry, which behaves much like a button widget. When a command entry is invoked, a Tcl command is executed. The Tcl command is specified with the −command option.

**SEPARATOR ENTRIES**

A separator is an entry that is displayed as a horizontal dividing line. A separator may not be activated or invoked, and it has no behavior other than its display appearance.

**CHECKBUTTON ENTRIES**

A checkbutton menu entry behaves much like a checkbutton widget. When it is invoked it toggles back and forth between the selected and deselected states. When the entry is selected, a particular value is stored in a particular global variable (as determined by the −onvalue and −variable options for the entry); when the entry is deselected another value (determined by the −offvalue option) is stored in the global variable. An indicator box is displayed to the left of the label in a checkbutton entry. If the entry is selected then the indicator's center is displayed in the color given by the −selectcolor option for the entry; otherwise the indicator's center is displayed in the background color for the menu. If a −command option is specified for a checkbutton entry, then its value is evaluated as a Tcl command each time the entry is invoked; this happens after toggling the entry's selected state.

**RADIOBUTTON ENTRIES**

A radiobutton menu entry behaves much like a radiobutton widget. Radiobutton entries are organized in groups of which only one entry may be selected at a time. Whenever a particular entry becomes selected it stores a particular value into a particular global variable (as determined by the −value and −variable options for the entry). This action causes any previously–selected entry in the same group to deselect itself. Once an entry has become selected, any change to the entry's associated variable will cause the entry to deselect itself. Grouping of radiobutton entries is determined by their associated variables: if two entries have the same
associated variable then they are in the same group. An indicator diamond is displayed to the left of the label in each radiobutton entry. If the entry is selected then the indicator's center is displayed in the color given by the \texttt{--selectcolor} option for the entry; otherwise the indicator's center is displayed in the background color for the menu. If a \texttt{--command} option is specified for a radiobutton entry, then its value is evaluated as a Tcl command each time the entry is invoked; this happens after selecting the entry.

\section*{Cascade Entries}

A cascade entry is one with an associated menu (determined by the \texttt{--menu} option). Cascade entries allow the construction of cascading menus. The \texttt{postcascade} widget command can be used to post and unpost the associated menu just next to of the cascade entry. The associated menu must be a child of the menu containing the cascade entry (this is needed in order for menu traversal to work correctly).

A cascade entry posts its associated menu by invoking a Tcl command of the form

\begin{verbatim}
menu post x y
\end{verbatim}

where \texttt{menu} is the path name of the associated menu, and \texttt{x} and \texttt{y} are the root–window coordinates of the upper–right corner of the cascade entry. On Unix, the lower–level menu is unposted by executing a Tcl command with the form

\begin{verbatim}
menu unpost
\end{verbatim}

where \texttt{menu} is the name of the associated menu. On other platforms, the platform's native code takes care of unposting the menu.

If a \texttt{--command} option is specified for a cascade entry then it is evaluated as a Tcl command whenever the entry is invoked. This is not supported on Windows.

\section*{ Tear–off Entries}

A tear–off entry appears at the top of the menu if enabled with the \texttt{tearOff} option. It is not like other menu entries in that it cannot be created with the \texttt{add} widget command and cannot be deleted with the \texttt{delete} widget command. When a tear–off entry is created it appears as a dashed line at the top of the menu. Under the default bindings, invoking the tear–off entry causes a torn–off copy to be made of the menu and all of its submenus.

\section*{ Menubars}

Any menu can be set as a menubar for a toplevel window (see \texttt{toplevel} command for syntax). On the Macintosh, whenever the toplevel is in front, this menu's cascade items will appear in the menubar across the top of the main monitor. On Windows and Unix, this menu's items will be displayed in a menubar across the top of the window. These menus will behave according to the interface guidelines of their platforms. For every menu set as a menubar, a clone menu is made. See the \texttt{CLONES} section for more information.
As noted, menubars may behave differently on different platforms. One example of this concerns the handling of checkbuttons and radiobuttons within the menu. While it is permitted to put these menu elements on menubars, they may not be drawn with indicators on some platforms, due to system restrictions.

**SPECIAL MENUS IN MENUBARS**

Certain menus in a menubar will be treated specially. On the Macintosh, access to the special Apple and Help menus is provided. On Windows, access to the Windows System menu in each window is provided. On X Windows, a special right-justified help menu is provided. In all cases, these menus must be created with the command name of the menubar menu concatenated with the special name. So for a menubar named `.menubar`, on the Macintosh, the special menus would be `.menubar.apple` and `.menubar.help`; on Windows, the special menu would be `.menubar.system`; on X Windows, the help menu would be `.menubar.help`.

When Tk sees an Apple menu on the Macintosh, that menu's contents make up the first items of the Apple menu on the screen whenever the window containing the menubar is in front. The menu is the first one that the user sees and has a title which is an Apple logo. After all of the Tk-defined items, the menu will have a separator, followed by all of the items in the user's Apple Menu Items folder. Since the System uses a different menu definition procedure for the Apple menu than Tk uses for its menus, and the system APIs do not fully support everything Tk tries to do, the menu item will only have its text displayed. No font attributes, images, bitmaps, or colors will be displayed. In addition, a menu with a tearoff item will have the tearoff item displayed as "(TearOff)".

When Tk sees a Help menu on the Macintosh, the menu's contents are appended to the standard help menu on the right of the user's menubar whenever the user's menubar is in front. The first items in the menu are provided by Apple. Similar to the Apple Menu, customization in this menu is limited to what the system provides.

When Tk sees a System menu on Windows, its items are appended to the system menu that the menubar is attached to. This menu has an icon representing a spacebar, and can be invoked with the mouse or by typing Alt+Spacebar. Due to limitations in the Windows API, any font changes, colors, images, bitmaps, or tearoff images will not appear in the system menu.

When Tk sees a Help menu on X Windows, the menu is moved to be last in the menubar and is right justified.

**CLONES**

When a menu is set as a menubar for a toplevel window, or when a menu is torn off, a clone of the menu is made. This clone is a menu widget in its own right, but it is a child of the original. Changes in the configuration of the original are reflected in the clone. Additionally, any cascades that are pointed to are also cloned so that menu traversal will work right. Clones are destroyed when either the tearoff or menubar goes away, or when the original menu is destroyed.
WIDGET COMMAND

The **menu** command creates a new Tcl command whose name is *pathName*. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

*Option* and the *args* determine the exact behavior of the command.

Many of the widget commands for a menu take as one argument an indicator of which entry of the menu to operate on. These indicators are called *indexes* and may be specified in any of the following forms:

- **number**
  - Specifies the entry numerically, where 0 corresponds to the top–most entry of the menu, 1 to the entry below it, and so on.
- **active**
  - Indicates the entry that is currently active. If no entry is active then this form is equivalent to *none*. This form may not be abbreviated.
- **end**
  - Indicates the bottommost entry in the menu. If there are no entries in the menu then this form is equivalent to *none*. This form may not be abbreviated.
- **last**
  - Same as *end*.
- **none**
  - Indicates ``no entry at all''; this is used most commonly with the *activate* option to deactivate all the entries in the menu. In most cases the specification of *none* causes nothing to happen in the widget command. This form may not be abbreviated.
- **@number**
  - In this form, *number* is treated as a y–coordinate in the menu's window; the entry closest to that y–coordinate is used. For example, ``@0'' indicates the top–most entry in the window.
- **pattern**
  - If the index doesn't satisfy one of the above forms then this form is used. *Pattern* is pattern–matched against the label of each entry in the menu, in order from the top down, until a matching entry is found. The rules of *Tcl StringMatch* are used.

The following widget commands are possible for menu widgets:

```
pathName activate index
```

Change the state of the entry indicated by *index* to *active* and redisplay it using its active colors. Any previously–active entry is deactivated. If *index* is specified as *none*, or if the specified entry is disabled, then the menu ends up with no active entry. Returns an empty string.

```
pathName add type ?option value option value ...?
```

Add a new entry to the bottom of the menu. The new entry's type is given by *type* and must be one of *cascade*, *checkbutton*, *command*, *radiobutton*, or *separator*, or a unique abbreviation of one of the above. If additional arguments are present, they specify any of the following options:
−activebackground value
Specifies a background color to use for displaying this entry when it is active. If this option is specified as an empty string (the default), then the activeBackground option for the overall menu is used. If the tk_strictMotif variable has been set to request strict Motif compliance, then this option is ignored and the −background option is used in its place. This option is not available for separator or tear-off entries.

−activeforeground value
Specifies a foreground color to use for displaying this entry when it is active. If this option is specified as an empty string (the default), then the activeForeground option for the overall menu is used. This option is not available for separator or tear-off entries.

−accelerator value
Specifies a string to display at the right side of the menu entry. Normally describes an accelerator keystroke sequence that may be typed to invoke the same function as the menu entry. This option is not available for separator or tear-off entries.

−background value
Specifies a background color to use for displaying this entry when it is in the normal state (neither active nor disabled). If this option is specified as an empty string (the default), then the background option for the overall menu is used. This option is not available for separator or tear-off entries.

−bitmap value
Specifies a bitmap to display in the menu instead of a textual label, in any of the forms accepted by Tk_GetBitmap. This option overrides the −label option (as controlled by the −compound option) but may be reset to an empty string to enable a textual label to be displayed. If a −image option has been specified, it overrides −bitmap. This option is not available for separator or tear-off entries.

−columnbreak value
When this option is zero, the entry appears below the previous entry. When this option is one, the entry appears at the top of a new column in the menu.

−command value
Specifies a Tcl command to execute when the menu entry is invoked. Not available for separator or tear-off entries.

−compound value
Specifies whether the menu entry should display both an image and text, and if so, where the image should be placed relative to the text. Valid values for this option are bottom, center, left, none, right and top. The default value is none, meaning that the button will display either an image or text, depending on the values of the −image and −bitmap options.

−font value
Specifies the font to use when drawing the label or accelerator string in this entry. If this option is specified as an empty string (the default) then the font option for the overall menu is used. This option is not available for separator or tear-off entries.

−foreground value
Specifies a foreground color to use for displaying this entry when it is in the normal state (neither active nor disabled). If this option is specified as an empty string (the default), then the foreground option for the overall menu is used. This option is not available for separator or tear-off entries.
−hidemargin value
Specifies whether the standard margins should be drawn for this menu entry. This is useful when creating palette with images in them, i.e., color palettes, pattern palettes, etc. 1 indicates that the margin for the entry is hidden; 0 means that the margin is used.

−image value
Specifies an image to display in the menu instead of a text string or bitmap. The image must have been created by some previous invocation of image create. This option overrides the −label and −bitmap options (as controlled by the −compound option) but may be reset to an empty string to enable a textual or bitmap label to be displayed. This option is not available for separator or tear-off entries.

−indicatoron value
Available only for checkbutton and radiobutton entries. Value is a boolean that determines whether or not the indicator should be displayed.

−label value
Specifies a string to display as an identifying label in the menu entry. Not available for separator or tear-off entries.

−menu value
Available only for cascade entries. Specifies the path name of the submenu associated with this entry. The submenu must be a child of the menu.

−offvalue value
Available only for checkbutton entries. Specifies the value to store in the entry's associated variable when the entry is deselected.

−onvalue value
Available only for checkbutton entries. Specifies the value to store in the entry's associated variable when the entry is selected.

−selectcolor value
Available only for checkbutton and radiobutton entries. Specifies the color to display in the indicator when the entry is selected. If the value is an empty string (the default) then the selectColor option for the menu determines the indicator color.

−selectimage value
Available only for checkbutton and radiobutton entries. Specifies an image to display in the entry (in place of the −image option) when it is selected. Value is the name of an image, which must have been created by some previous invocation of image create. This option is ignored unless the −image option has been specified.

−state value
Specifies one of three states for the entry: normal, active, or disabled. In normal state the entry is displayed using the foreground option for the menu and the background option from the entry or the menu. The active state is typically used when the pointer is over the entry. In active state the entry is displayed using the activeForeground option for the menu along with the activebackground option from the entry. Disabled state means that the entry should be insensitive: the default bindings will refuse to activate or invoke the entry. In this state the entry is displayed according to the disabledForeground option for the menu and the background option from the entry. This option is not available for separator entries.

−underline value
Specifies the integer index of a character to underline in the entry. This option is also queried
by the default bindings and used to implement keyboard traversal. 0 corresponds to the first character of the text displayed in the entry, 1 to the next character, and so on. If a bitmap or image is displayed in the entry then this option is ignored. This option is not available for separator or tear−off entries.

−value value
   Available only for radiobutton entries. Specifies the value to store in the entry’s associated variable when the entry is selected. If an empty string is specified, then the −label option for the entry as the value to store in the variable.

−variable value
   Available only for checkbutton and radiobutton entries. Specifies the name of a global value to set when the entry is selected. For checkbutton entries the variable is also set when the entry is deselected. For radiobutton entries, changing the variable causes the currently−selected entry to deselect itself.

The add widget command returns an empty string.

pathName cget option
   Returns the current value of the configuration option given by option. Option may have any of the values accepted by the menu command.

pathName clone newPathName ?cloneType?
   Makes a clone of the current menu named newPathName. This clone is a menu in its own right, but any changes to the clone are propagated to the original menu and vice versa. cloneType can be normal, menubar, or tearoff. Should not normally be called outside of the Tk library. See the CLONES section for more information.

pathName configure ?option? ?value option value ...?
   Query or modify the configuration options of the widget. If no option is specified, returns a list describing all of the available options for pathName (see Tk_ConfigureInfo for information on the format of this list). If option is specified with no value, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no option is specified). If one or more option−value pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. Option may have any of the values accepted by the menu command.

pathName delete index1 ?index2?
   Delete all of the menu entries between index1 and index2 inclusive. If index2 is omitted then it defaults to index1. Attempts to delete a tear−off menu entry are ignored (instead, you should change the tearOff option to remove the tear−off entry).

pathName entrycget index option
   Returns the current value of a configuration option for the entry given by index. Option may have any of the values accepted by the add widget command.

pathName entryconfigure index ?options?
   This command is similar to the configure command, except that it applies to the options for an individual entry, whereas configure applies to the options for the menu as a whole. Options may have any of the values accepted by the add widget command. If options are specified, options are modified as indicated in the command and the command returns an empty string. If no options are specified, returns a list describing the current options for entry index (see Tk_ConfigureInfo for information on the format of this list).

pathName index index
Returns the numerical index corresponding to \textit{index}, or \textbf{none} if \textit{index} was specified as \textbf{none}.

\texttt{pathName insert index type \?option value option value ...\?}

Same as the \texttt{add} widget command except that it inserts the new entry just before the entry given by \textit{index}, instead of appending to the end of the menu. The \textit{type}, \textit{option}, and \textit{value} arguments have the same interpretation as for the \texttt{add} widget command. It is not possible to insert new menu entries before the tear-off entry, if the menu has one.

\texttt{pathName invoke index}

Invoke the action of the menu entry. See the sections on the individual entries above for details on what happens. If the menu entry is disabled then nothing happens. If the entry has a command associated with it then the result of that command is returned as the result of the \texttt{invoke} widget command. Otherwise the result is an empty string. Note: invoking a menu entry does not automatically unpost the menu; the default bindings normally take care of this before invoking the \texttt{invoke} widget command.

\texttt{pathName post x y}

Arrange for the menu to be displayed on the screen at the root–window coordinates given by \textit{x} and \textit{y}. These coordinates are adjusted if necessary to guarantee that the entire menu is visible on the screen. This command normally returns an empty string. If the \texttt{postCommand} option has been specified, then its value is executed as a Tcl script before posting the menu and the result of that script is returned as the result of the \texttt{post} widget command. If an error returns while executing the command, then the error is returned without posting the menu.

\texttt{pathName postcascade index}

Posts the submenu associated with the cascade entry given by \textit{index}, and unposts any previously posted submenu. If \textit{index} doesn't correspond to a cascade entry, or if \texttt{pathName} isn't posted, the command has no effect except to unpost any currently posted submenu.

\texttt{pathName type index}

Returns the type of the menu entry given by \textit{index}. This is the \textit{type} argument passed to the \texttt{add} widget command when the entry was created, such as \textbf{command} or \textbf{separator}, or \textbf{tearoff} for a tear–off entry.

\texttt{pathName unpost}

Unmap the window so that it is no longer displayed. If a lower–level cascaded menu is posted, unpost that menu. Returns an empty string. This subcommand does not work on Windows and the Macintosh, as those platforms have their own way of unposting menus.

\texttt{pathName yposition index}

Returns a decimal string giving the y–coordinate within the menu window of the topmost pixel in the entry specified by \textit{index}.

\textbf{MENU CONFIGURATIONS}

The default bindings support four different ways of using menus:

\textbf{Pulldown Menus in Menubar}

This is the most command case. You create a menu widget that will become the menu bar. You then add cascade entries to this menu, specifying the pull down menus you wish to use in your menu bar. You then create all of the pulldowns. Once you have done this, specify the menu using the \texttt{–menu} option of the toplevel's widget command. See the \texttt{toplevel} manual entry for details.


**Pulldown Menus in Menu Buttons**

This is the compatible way to do menu bars. You create one menubutton widget for each top-level menu, and typically you arrange a series of menubuttons in a row in a menubar window. You also create the top-level menus and any cascaded submenus, and tie them together with `-menu` options in menubuttons and cascade menu entries. The top-level menu must be a child of the menubutton, and each submenu must be a child of the menu that refers to it. Once you have done this, the default bindings will allow users to traverse and invoke the tree of menus via its menubutton; see the `menubutton` manual entry for details.

**Popup Menus**

Popup menus typically post in response to a mouse button press or keystroke. You create the popup menus and any cascaded submenus, then you call the `tk_popup` procedure at the appropriate time to post the top-level menu.

**Option Menus**

An option menu consists of a menubutton with an associated menu that allows you to select one of several values. The current value is displayed in the menubutton and is also stored in a global variable. Use the `tk_optionMenu` procedure to create option menubuttons and their menus.

**Torn-off Menus**

You create a torn-off menu by invoking the tear-off entry at the top of an existing menu. The default bindings will create a new menu that is a copy of the original menu and leave it permanently posted as a top-level window. The torn-off menu behaves just the same as the original menu.

**DEFAULT BINDINGS**

Tk automatically creates class bindings for menus that give them the following default behavior:

1. When the mouse enters a menu, the entry underneath the mouse cursor activates; as the mouse moves around the menu, the active entry changes to track the mouse.
2. When the mouse leaves a menu all of the entries in the menu deactivate, except in the special case where the mouse moves from a menu to a cascaded submenu.
3. When a button is released over a menu, the active entry (if any) is invoked. The menu also unposts unless it is a torn-off menu.
4. The Space and Return keys invoke the active entry and unpost the menu.
5. If any of the entries in a menu have letters underlined with the `-underline` option, then pressing one of the underlined letters (or its upper-case or lower-case equivalent) invokes that entry and unposts the menu.
6. The Escape key aborts a menu selection in progress without invoking any entry. It also unposts the menu unless it is a torn-off menu.
7. The Up and Down keys activate the next higher or lower entry in the menu. When one end of the...
menu is reached, the active entry wraps around to the other end.

[8] The Left key moves to the next menu to the left. If the current menu is a cascaded submenu, then the submenu is unposted and the current menu entry becomes the cascade entry in the parent. If the current menu is a top–level menu posted from a menubutton, then the current menubutton is unposted and the next menubutton to the left is posted. Otherwise the key has no effect. The left–right order of menubuttons is determined by their stacking order: Tk assumes that the lowest menubutton (which by default is the first one created) is on the left.

[9] The Right key moves to the next menu to the right. If the current entry is a cascade entry, then the submenu is posted and the current menu entry becomes the first entry in the submenu. Otherwise, if the current menu was posted from a menubutton, then the current menubutton is unposted and the next menubutton to the right is posted.

Disabled menu entries are non–responsive: they don't activate and they ignore mouse button presses and releases.

Several of the bindings make use of the command `tk_menuSetFocus`. It saves the current focus and sets the focus to its `pathName` argument, which is a menu widget.

The behavior of menus can be changed by defining new bindings for individual widgets or by redefining the class bindings.

**BUGS**

At present it isn't possible to use the option database to specify values for the options to individual entries.

**KEYWORDS**

```
menu, widget
```
menubutton

NAME

menubutton – Create and manipulate menubutton widgets

SYNOPSIS

`menubutton pathName ?options?`

STANDARD OPTIONS

- `−activebackground`, `activeBackground`, `Foreground`
- `−activeforeground`, `activeForeground`, `Background`
- `−anchor`, `anchor`, `Anchor`
- `−background or −bg`, `background`, `Background`
- `−bitmap`, `bitmap`, `Bitmap`
- `−borderwidth or −bd`, `borderWidth`, `BorderWidth`
- `−compound`, `compound`, `Compound`
- `−cursor`, `cursor`, `Cursor`
- `−disabledforeground`, `disabledForeground`, `DisabledForeground`
- `−font`, `font`, `Font`
- `−foreground or −fg`, `foreground`, `Foreground`
- `−highlightbackground`, `highlightBackground`, `HighlightBackground`
- `−highlightcolor`, `highlightColor`, `HighlightColor`
- `−highlightthickness`, `highlightThickness`, `HighlightThickness`
- `−image`, `image`, `Image`
- `−justify`, `justify`, `Justify`
- `−padX`, `padX`, `Pad`
- `−padY`, `padY`, `Pad`
- `−relief`, `relief`, `Relief`
- `−takefocus`, `takeFocus`, `TakeFocus`
- `−text`, `text`, `Text`
- `−textvariable`, `textVariable`, `Variable`
- `−underline`, `underline`, `Underline`
- `−wraplength`, `wrapLength`, `WrapLength`

WIDGET–SPECIFIC OPTIONS

- `−direction`, `direction`, `Height`
- `−height`, `height`, `Height`
- `−indicatoron`, `indicatorOn`, `IndicatorOn`
- `−menu`, `menu`, `MenuName`
- `−state`, `state`, `State`
- `−width`, `width`, `Width`

INTRODUCTION

WIDGET COMMAND

`pathName cget option`

`pathName configure ?option? ?value option value ...?`

DEFAULT BINDINGS
KEYWORDS

NAME

menubutton – Create and manipulate menubutton widgets

SYNOPSIS

menubutton pathName ?options?

STANDARD OPTIONS

−activebackground, activeBackground, Foreground
−activeforeground, activeForeground, Background
−anchor, anchor, Anchor
−background or −bg, background, Background
−bitmap, bitmap, Bitmap
−borderwidth or −bd, borderWidth, BorderWidth
−compound, compound, Compound
−cursor, cursor, Cursor
−disabledforeground, disabledForeground, DisabledForeground
−font, font, Font
−foreground or −fg, foreground, Foreground
−highlightbackground, highlightBackground, HighlightBackground
−highlightcolor, highlightColor, HighlightColor
−highlightthickness, highlightThickness, HighlightThickness
−image, image, Image
−justify, justify, Justify
−padx, padX, Pad
−pady, padY, Pad
−relief, relief, Relief
−takefocus, takeFocus, TakeFocus
−text, text, Text
−textvariable, textVariable, Variable
−underline, underline, Underline
−wraplength, wrapLength, WrapLength

WIDGET−SPECIFIC OPTIONS

Command−Line Name: −direction
Database Name: direction
Database Class: Height

Specifies where the menu is going to be popup up. above tries to pop the menu above the
menubutton. below tries to pop the menu below the menubutton. left tries to pop the menu to the left

of the menubutton. right tries to pop the menu to the right of the menu button. flush pops the menu directly over the menubutton. In the case of above or below, the direction will be reversed if the menu would show offscreen.

Command–Line Name: \-height
Database Name: height
Database Class: Height
Specifications a desired height for the menubutton. If an image or bitmap is being displayed in the menubutton then the value is in screen units (i.e. any of the forms acceptable to Tk_GetPixels); for text it is in lines of text. If this option isn't specified, the menubutton's desired height is computed from the size of the image or bitmap or text being displayed in it.

Command–Line Name: \-indicatoron
Database Name: indicatorOn
Database Class: IndicatorOn
The value must be a proper boolean value. If it is true then a small indicator rectangle will be displayed on the right side of the menubutton and the default menu bindings will treat this as an option menubutton. If false then no indicator will be displayed.

Command–Line Name: \-menu
Database Name: menu
Database Class: MenuName
Specifies the path name of the menu associated with this menubutton. The menu must be a child of the menubutton.

Command–Line Name: \-state
Database Name: state
Database Class: State
Specifies one of three states for the menubutton: normal, active, or disabled. In normal state the menubutton is displayed using the foreground and background options. The active state is typically used when the pointer is over the menubutton. In active state the menubutton is displayed using the activeForeground and activeBackground options. Disabled state means that the menubutton should be insensitive: the default bindings will refuse to activate the widget and will ignore mouse button presses. In this state the disabledForeground and background options determine how the button is displayed.

Command–Line Name: \-width
Database Name: width
Database Class: Width
Specifies a desired width for the menubutton. If an image or bitmap is being displayed in the menubutton then the value is in screen units (i.e. any of the forms acceptable to Tk_GetPixels); for text it is in characters. If this option isn't specified, the menubutton's desired width is computed from the size of the image or bitmap or text being displayed in it.

INTRODUCTION

The menubutton command creates a new window (given by the pathName argument) and makes it into a menubutton widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the menubutton such as its colors, font, text, and initial relief. The menubutton command returns its pathName argument. At the time this command is invoked, there must not
exist a window named `pathName`, but `pathName`'s parent must exist.

A menubutton is a widget that displays a textual string, bitmap, or image and is associated with a menu widget. If text is displayed, it must all be in a single font, but it can occupy multiple lines on the screen (if it contains newlines or if wrapping occurs because of the `wrapLength` option) and one of the characters may optionally be underlined using the `underline` option. In normal usage, pressing mouse button 1 over the menubutton causes the associated menu to be posted just underneath the menubutton. If the mouse is moved over the menu before releasing the mouse button, the button release causes the underlying menu entry to be invoked. When the button is released, the menu is unposted.

Menubuttons are typically organized into groups called menu bars that allow scanning: if the mouse button is pressed over one menubutton (causing it to post its menu) and the mouse is moved over another menubutton in the same menu bar without releasing the mouse button, then the menu of the first menubutton is unposted and the menu of the new menubutton is posted instead.

There are several interactions between menubuttons and menus; see the `menu` manual entry for information on various menu configurations, such as pulldown menus and option menus.

**WIDGET COMMAND**

The `menubutton` command creates a new Tcl command whose name is `pathName`. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...? 
```

`Option` and the `args` determine the exact behavior of the command. The following commands are possible for menubutton widgets:

```
pathName cget option

Returns the current value of the configuration option given by `option`. `Option` may have any of the values accepted by the `menubutton` command.

pathName configure ?option? ?value option value ...?

Query or modify the configuration options of the widget. If no `option` is specified, returns a list describing all of the available options for `pathName` (see `Tk_ConfigureInfo` for information on the format of this list). If `option` is specified with no value, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no `option` is specified). If one or more `option=value` pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. `Option` may have any of the values accepted by the `menubutton` command.
```

**DEFAULT BINDINGS**

Tk automatically creates class bindings for menubuttons that give them the following default behavior:

[1]
A menubutton activates whenever the mouse passes over it and deactivates whenever the mouse leaves it.

[2] Pressing mouse button 1 over a menubutton posts the menubutton: its relief changes to raised and its associated menu is posted under the menubutton. If the mouse is dragged down into the menu with the button still down, and if the mouse button is then released over an entry in the menu, the menubutton is unposted and the menu entry is invoked.

[3] If button 1 is pressed over a menubutton and then released over that menubutton, the menubutton stays posted: you can still move the mouse over the menu and click button 1 on an entry to invoke it. Once a menu entry has been invoked, the menubutton unposts itself.

[4] If button 1 is pressed over a menubutton and then dragged over some other menubutton, the original menubutton unposts itself and the new menubutton posts.

[5] If button 1 is pressed over a menubutton and released outside any menubutton or menu, the menubutton unposts without invoking any menu entry.

[6] When a menubutton is posted, its associated menu claims the input focus to allow keyboard traversal of the menu and its submenus. See the menu manual entry for details on these bindings.

[7] If the underline option has been specified for a menubutton then keyboard traversal may be used to post the menubutton: Alt+x, where x is the underlined character (or its lower-case or upper-case equivalent), may be typed in any window under the menubutton's toplevel to post the menubutton.

[8] The F10 key may be typed in any window to post the first menubutton under its toplevel window that isn't disabled.

[9] If a menubutton has the input focus, the space and return keys post the menubutton.

If the menubutton's state is disabled then none of the above actions occur: the menubutton is completely non-responsive.

The behavior of menubuttons can be changed by defining new bindings for individual widgets or by redefining the class bindings.

KEYWORDS

menubutton, widget

Copyright ' 1990-1994 The Regents of the University of California.
Copyright ' 1994-1997 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

message – Create and manipulate message widgets

SYNOPSIS

message pathName ?options?

STANDARD OPTIONS

- anchor, anchor, Anchor
- background or -bg, background, Background
- borderwidth or -bd, borderWidth, BorderWidth
- cursor, cursor, Cursor
- font, font, Font
- foreground or -fg, foreground, Foreground
- highlightbackground, highlightBackground, HighlightBackground
- highlightcolor, highlightColor, HighlightColor
- highlightthickness, highlightThickness, HighlightThickness
- padx, padX, Pad
- pady, padY, Pad
- relief, relief, Relief
- takefocus, takeFocus, TakeFocus
- text, text, Text
- textvariable, textVariable, Variable
- width

WIDGET–SPECIFIC OPTIONS

- aspect, aspect, Aspect
- justify, justify, Justify
- width, width, Width

DESCRIPTION

WIDGET COMMAND

pathName cget option

pathName configure ?option? ?value option value ...?

DEFAULT BINDINGS

BUGS

KEYWORDS

NAME

message – Create and manipulate message widgets
SYNOPSIS

message pathName ?options?

STANDARD OPTIONS

 anchor, anchor, Anchor
 background or −bg, background, Background
 borderwidth or −bd, borderWidth, BorderWidth
 cursor, cursor, Cursor
 font, font, Font
 foreground or −fg, foreground, Foreground
 highlightbackground, highlightBackground, HighlightBackground
 highlightcolor, highlightColor, HighlightColor
 highlightthickness, highlightThickness, HighlightThickness
 padx, padX, Pad
 pady, padY, Pad
 relief, relief, Relief
 takefocus, takeFocus, TakeFocus
 text, text, Text
 textvariable, textVariable, Variable
 width

WIDGET–SPECIFIC OPTIONS

Command–Line Name: −aspect
Database Name: aspect
Database Class: Aspect
 Specifies a non–negative integer value indicating desired aspect ratio for the text. The aspect ratio is specified as 100*width/height. 100 means the text should be as wide as it is tall, 200 means the text should be twice as wide as it is tall, 50 means the text should be twice as tall as it is wide, and so on. Used to choose line length for text if width option isn't specified. Defaults to 150.

Command–Line Name: −justify
Database Name: justify
Database Class: Justify
 Specifies how to justify lines of text. Must be one of left, center, or right. Defaults to left. This option works together with the anchor, aspect, padX, padY, and width options to provide a variety of arrangements of the text within the window. The aspect and width options determine the amount of screen space needed to display the text. The anchor, padX, and padY options determine where this rectangular area is displayed within the widget's window, and the justify option determines how each line is displayed within that rectangular region. For example, suppose anchor is e and justify is left, and that the message window is much larger than needed for the text. The text will be displayed so that the left edges of all the lines line up and the right edge of the longest line is padX from the right side of the window; the entire text block will be centered in the vertical span of the window.

Command–Line Name: −width
**Description**

The **message** command creates a new window (given by the *pathName* argument) and makes it into a message widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the message such as its colors, font, text, and initial relief. The **message** command returns its *pathName* argument. At the time this command is invoked, there must not exist a window named *pathName*, but *pathName*'s parent must exist.

A message is a widget that displays a textual string. A message widget has three special features. First, it breaks up its string into lines in order to produce a given aspect ratio for the window. The line breaks are chosen at word boundaries wherever possible (if not even a single word would fit on a line, then the word will be split across lines). Newline characters in the string will force line breaks; they can be used, for example, to leave blank lines in the display.

The second feature of a message widget is justification. The text may be displayed left-justified (each line starts at the left side of the window), centered on a line-by-line basis, or right-justified (each line ends at the right side of the window).

The third feature of a message widget is that it handles control characters and non-printing characters specially. Tab characters are replaced with enough blank space to line up on the next 8-character boundary. Newlines cause line breaks. Other control characters (ASCII code less than 0x20) and characters not defined in the font are displayed as a four-character sequence \xhh where hh is the two-digit hexadecimal number corresponding to the character. In the unusual case where the font doesn't contain all of the characters in `\0123456789abcdef\x` then control characters and undefined characters are not displayed at all.

**Widget Command**

The **message** command creates a new Tcl command whose name is *pathName*. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

*Option* and the *args* determine the exact behavior of the command. The following commands are possible for message widgets:

```
pathName cget option
```

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **message** command.
pathName configure ?option? ?value option value ...?
Query or modify the configuration options of the widget. If no option is specified, returns a list describing all of the available options for pathName (see Tk_ConfigureInfo for information on the format of this list). If option is specified with no value, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no option is specified). If one or more option=value pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. Option may have any of the values accepted by the message command.

DEFAULT BINDINGS

When a new message is created, it has no default event bindings: messages are intended for output purposes only.

BUGS

Tabs don't work very well with text that is centered or right-justified. The most common result is that the line is justified wrong.

KEYWORDS

message, widget

Copyright © 1990–1994 The Regents of the University of California.
Copyright © 1994–1996 Sun Microsystems, Inc.
**NAME**

option – Add/retrieve window options to/from the option database

**SYNOPSIS**

option add pattern value ?priority?
option clear
option get window name class
option readfile fileName ?priority?

**DESCRIPTION**

The option command allows you to add entries to the Tk option database or to retrieve options from the database. The add form of the command adds a new option to the database. Pattern contains the option being specified, and consists of names and/or classes separated by asterisks or dots, in the usual X format. Value contains a text string to associate with pattern; this is the value that will be returned in calls to Tcl\_GetOption or by invocations of the option get command. If priority is specified, it indicates the priority level for this option (see below for legal values); it defaults to interactive. This command always returns an empty string.

The option clear command clears the option database. Default options (from the RESOURCE\_MANAGER property or the .Xdefaults file) will be reloaded automatically the next time an option is added to the database or removed from it. This command always returns an empty string.
The **option get** command returns the value of the option specified for *window* under *name* and *class*. If several entries in the option database match *window*, *name*, and *class*, then the command returns whichever was created with highest *priority* level. If there are several matching entries at the same priority level, then it returns whichever entry was most recently entered into the option database. If there are no matching entries, then the empty string is returned.

The **readfile** form of the command reads *fileName*, which should have the standard format for an X resource database such as `.Xdefaults`, and adds all the options specified in that file to the option database. If *priority* is specified, it indicates the priority level at which to enter the options; *priority* defaults to **interactive**.

The *priority* arguments to the **option** command are normally specified symbolically using one of the following values:

- **widgetDefault**: Level 20. Used for default values hard–coded into widgets.
- **startupFile**: Level 40. Used for options specified in application–specific startup files.
- **userDefault**: Level 60. Used for options specified in user–specific defaults files, such as `.Xdefaults`, resource databases loaded into the X server, or user–specific startup files.
- **interactive**: Level 80. Used for options specified interactively after the application starts running. If *priority* isn't specified, it defaults to this level.

Any of the above keywords may be abbreviated. In addition, priorities may be specified numerically using integers between 0 and 100, inclusive. The numeric form is probably a bad idea except for new priority levels other than the ones given above.

**EXAMPLES**

Instruct every button in the application to have red text on it unless explicitly overridden:

```
option add *button.foreground red startupFile
```

Allow users to control what happens in an entry widget when the Return key is pressed by specifying a script in the option database and add a default option for that which rings the bell:

```
entry .e
bind .e <Return> [option get .e returnCommand Command]
option add *.e.returnCommand bell widgetDefault
```

**KEYWORDS**

database, option, priority, retrieve

---

*Copyright © 1990 The Regents of the University of California.*
NAME

options – Standard options supported by widgets

DESCRIPTION

- activebackground, activeBackground, Foreground
- activeborderwidth, activeBorderWidth, BorderWidth
- activeforeground, activeForeground, Background
- anchor, anchor, Anchor
- background or -bg, background, Background
- bitmap, bitmap, Bitmap
- borderwidth or -bd, borderWidth, BorderWidth
- cursor, cursor, Cursor
- compound, compound, Compound
- disabledforeground, disabledForeground, DisabledForeground
- exportselection, exportSelection, ExportSelection
- font, font, Font
- foreground or -fg, foreground, Foreground
- highlightbackground, highlightBackground, HighlightBackground
- highlightcolor, highlightColor, HighlightColor
- highlightthickness, highlightThickness, HighlightThickness
- image, image, Image
- insertbackground, insertBackground, Foreground
- insertborderwidth, insertBorderWidth, BorderWidth
- insertofftime, insertOffTime, OffTime
- insertontime, insertOnTime, OnTime
- insertwidth, insertWidth, InsertWidth
- jump, jump, Jump
- justify, justify, Justify
- orient, orient, Orient
- padx, padX, Pad
- pady, padY, Pad
- relief, relief, Relief
- replatedelay, repeatDelay, RepeatDelay
- repeatinterval, repeatInterval, RepeatInterval
- selectbackground, selectBackground, Foreground
- selectborderwidth, selectBorderWidth, BorderWidth
- selectforeground, selectForeground, Background
- setgrid, setGrid, SetGrid
- takefocus, takeFocus, TakeFocus
- text, text, Text
- textvariable, textVariable, Variable
- troughcolor, troughColor, Background

options 553
NAME

options – Standard options supported by widgets

DESCRIPTION

This manual entry describes the common configuration options supported by widgets in the Tk toolkit. Every widget does not necessarily support every option (see the manual entries for individual widgets for a list of the standard options supported by that widget), but if a widget does support an option with one of the names listed below, then the option has exactly the effect described below.

In the descriptions below, "Command−Line Name" refers to the switch used in class commands and configure widget commands to set this value. For example, if an option's command−line switch is −foreground and there exists a widget .a.b.c, then the command

```
.a.b.c configure −foreground black
```

may be used to specify the value black for the option in the widget .a.b.c. Command−line switches may be abbreviated, as long as the abbreviation is unambiguous. "Database Name" refers to the option's name in the option database (e.g. in .Xdefaults files). "Database Class" refers to the option's class value in the option database.

Command−Line Name: −activebackground
Database Name: activeBackground
Database Class: Foreground

Specifies background color to use when drawing active elements. An element (a widget or portion of a widget) is active if the mouse cursor is positioned over the element and pressing a mouse button will cause some action to occur. If strict Motif compliance has been requested by setting the tk_strictMotif variable, this option will normally be ignored; the normal background color will be used instead. For some elements on Windows and Macintosh systems, the active color will only be used while mouse button 1 is pressed over the element.

Command−Line Name: −activeborderwidth
Database Name: activeBorderWidth
Database Class: BorderWidth

Specifies a non–negative value indicating the width of the 3–D border drawn around active elements. See above for definition of active elements. The value may have any of the forms acceptable to Tk_GetPixels. This option is typically only available in widgets displaying more than one element at a time (e.g. menus but not buttons).
Command–Line Name: −activeforeground
Database Name: activeForeground
Database Class: Background
   Specifies foreground color to use when drawing active elements. See above for definition of active elements.

Command–Line Name: −anchor
Database Name: anchor
Database Class: Anchor
   Specifies how the information in a widget (e.g. text or a bitmap) is to be displayed in the widget. Must be one of the values n, ne, e, se, s, sw, w, nw, or center. For example, nw means display the information such that its top–left corner is at the top–left corner of the widget.

Command–Line Name: −background or −bg
Database Name: background
Database Class: Background
   Specifies the normal background color to use when displaying the widget.

Command–Line Name: −bitmap
Database Name: bitmap
Database Class: Bitmap
   Specifies a bitmap to display in the widget, in any of the forms acceptable to Tk_GetBitmap. The exact way in which the bitmap is displayed may be affected by other options such as anchor or justify. Typically, if this option is specified then it overrides other options that specify a textual value to display in the widget but this is controlled by the compound option; the bitmap option may be reset to an empty string to re–enable a text display. In widgets that support both bitmap and image options, image will usually override bitmap.

Command–Line Name: −borderwidth or −bd
Database Name: borderWidth
Database Class: BorderWidth
   Specifies a non–negative value indicating the width of the 3–D border to draw around the outside of the widget (if such a border is being drawn; the relief option typically determines this). The value may also be used when drawing 3–D effects in the interior of the widget. The value may have any of the forms acceptable to Tk_GetPixels.

Command–Line Name: −cursor
Database Name: cursor
Database Class: Cursor
   Specifies the mouse cursor to be used for the widget. The value may have any of the forms acceptable to Tk_GetCursor.

Command–Line Name: −compound
Database Name: compound
Database Class: Compound
   Specifies if the widget should display text and bitmaps/images at the same time, and if so, where the bitmap/image should be placed relative to the text. Must be one of the values none, bottom, top, left, right, or center. For example, the (default) value none specifies that the bitmap or image should (if defined) be displayed instead of the text, the value left specifies that the bitmap or image should be displayed to the left of the text, and the value center specifies that the bitmap or image should be displayed on top of the text.
Command—Line Name: −disabledforeground
Database Name: disabledForeground
Database Class: DisabledForeground
Specifies foreground color to use when drawing a disabled element. If the option is specified as an empty string (which is typically the case on monochrome displays), disabled elements are drawn with the normal foreground color but they are dimmed by drawing them with a stippled fill pattern.

Command—Line Name: −exportselection
Database Name: exportSelection
Database Class: ExportSelection
Specifies whether or not a selection in the widget should also be the X selection. The value may have any of the forms accepted by Tcl_GetBoolean, such as true, false, 0, 1, yes, or no. If the selection is exported, then selecting in the widget deselects the current X selection, selecting outside the widget deselects any widget selection, and the widget will respond to selection retrieval requests when it has a selection. The default is usually for widgets to export selections.

Command—Line Name: −font
Database Name: font
Database Class: Font
Specifies the font to use when drawing text inside the widget. The value may have any of the forms accepted by Tk_GetFont.

Command—Line Name: −foreground or −fg
Database Name: foreground
Database Class: Foreground
Specifies the normal foreground color to use when displaying the widget.

Command—Line Name: −highlightbackground
Database Name: highlightBackground
Database Class: HighlightBackground
Specifies the color to display in the traversal highlight region when the widget does not have the input focus.

Command—Line Name: −highlightcolor
Database Name: highlightColor
Database Class: HighlightColor
Specifies the color to use for the traversal highlight rectangle that is drawn around the widget when it has the input focus.

Command—Line Name: −highlightthickness
Database Name: highlightThickness
Database Class: HighlightThickness
Specifies a non-negative value indicating the width of the highlight rectangle to draw around the outside of the widget when it has the input focus. The value may have any of the forms acceptable to Tk_GetPixels. If the value is zero, no focus highlight is drawn around the widget.

Command—Line Name: −image
Database Name: image
Database Class: Image
Specifies an image to display in the widget, which must have been created with the image create command. Typically, if the image option is specified then it overrides other options that specify a bitmap or textual value to display in the widget, though this is controlled by the compound option;
the image option may be reset to an empty string to re-enable a bitmap or text display.

**Command-Line Name:** −insertbackground
**Database Name:** insertBackground
**Database Class:** Foreground

Specifies the color to use as background in the area covered by the insertion cursor. This color will normally override either the normal background for the widget (or the selection background if the insertion cursor happens to fall in the selection).

**Command-Line Name:** −insertborderwidth
**Database Name:** insertBorderWidth
**Database Class:** BorderWidth

Specifies a non-negative value indicating the width of the 3-D border to draw around the insertion cursor. The value may have any of the forms acceptable to Tk_GetPixels.

**Command-Line Name:** −insertofftime
**Database Name:** insertOffTime
**Database Class:** OffTime

Specifies a non-negative integer value indicating the number of milliseconds the insertion cursor should remain "off" in each blink cycle. If this option is zero then the cursor doesn't blink: it is on all the time.

**Command-Line Name:** −insertontime
**Database Name:** insertOnTime
**Database Class:** OnTime

Specifies a non-negative integer value indicating the number of milliseconds the insertion cursor should remain "on" in each blink cycle.

**Command-Line Name:** −insertwidth
**Database Name:** insertWidth
**Database Class:** InsertWidth

Specifies a value indicating the total width of the insertion cursor. The value may have any of the forms acceptable to Tk_GetPixels. If a border has been specified for the insertion cursor (using the insertBorderWidth option), the border will be drawn inside the width specified by the insertWidth option.

**Command-Line Name:** −jump
**Database Name:** jump
**Database Class:** Jump

For widgets with a slider that can be dragged to adjust a value, such as scrollbars, this option determines when notifications are made about changes in the value. The option's value must be a boolean of the form accepted by Tcl_GetBoolean. If the value is false, updates are made continuously as the slider is dragged. If the value is true, updates are delayed until the mouse button is released to end the drag; at that point a single notification is made (the value "jumps" rather than changing smoothly).

**Command-Line Name:** −justify
**Database Name:** justify
**Database Class:** Justify

When there are multiple lines of text displayed in a widget, this option determines how the lines line up with each other. Must be one of left, center, or right. Left means that the lines' left edges all line up, center means that the lines' centers are aligned, and right means that the lines' right edges line up.
Command−Line Name: −orient
Database Name: orient
Database Class: Orient
For widgets that can lay themselves out with either a horizontal or vertical orientation, such as
scrollbars, this option specifies which orientation should be used. Must be either horizontal or
vertical or an abbreviation of one of these.

Command−Line Name: −padx
Database Name: padX
Database Class: Pad
Specifies a non−negative value indicating how much extra space to request for the widget in the
X−direction. The value may have any of the forms acceptable to Tk_GetPixels. When computing
how large a window it needs, the widget will add this amount to the width it would normally need (as
determined by the width of the things displayed in the widget); if the geometry manager can satisfy
this request, the widget will end up with extra internal space to the left and/or right of what it displays
inside. Most widgets only use this option for padding text: if they are displaying a bitmap or image,
then they usually ignore padding options.

Command−Line Name: −pady
Database Name: padY
Database Class: Pad
Specifies a non−negative value indicating how much extra space to request for the widget in the
Y−direction. The value may have any of the forms acceptable to Tk_GetPixels. When computing
how large a window it needs, the widget will add this amount to the height it would normally need (as
determined by the height of the things displayed in the widget); if the geometry manager can satisfy
this request, the widget will end up with extra internal space above and/or below what it displays
inside. Most widgets only use this option for padding text: if they are displaying a bitmap or image,
then they usually ignore padding options.

Command−Line Name: −relief
Database Name: relief
Database Class: Relief
Specifies the 3−D effect desired for the widget. Acceptable values are raised, sunken, flat, ridge,
solid, and groove. The value indicates how the interior of the widget should appear relative to its
exterior; for example, raised means the interior of the widget should appear to protrude from the
screen, relative to the exterior of the widget.

Command−Line Name: −repeatdelay
Database Name: repeatDelay
Database Class: RepeatDelay
Specifies the number of milliseconds a button or key must be held down before it begins to
auto−repeat. Used, for example, on the up− and down−arrows in scrollbars.

Command−Line Name: −repeatinterval
Database Name: repeatInterval
Database Class: RepeatInterval
Used in conjunction with repeatDelay: once auto−repeat begins, this option determines the number
of milliseconds between auto−repeats.

Command−Line Name: −selectbackground
Database Name: selectBackground
**Database Class: Foreground**

Specifies the background color to use when displaying selected items.

**Command-Line Name:** –selectborderwidth

**Database Name:** selectBorderWidth

**Database Class: BorderWidth**

Specifies a non-negative value indicating the width of the 3-D border to draw around selected items. The value may have any of the forms acceptable to `Tk_GetPixels`.

**Command-Line Name:** –selectforeground

**Database Name:** selectForeground

**Database Class: Background**

Specifies the foreground color to use when displaying selected items.

**Command-Line Name:** –setgrid

**Database Name:** setGrid

**Database Class: SetGrid**

Specifies a boolean value that determines whether this widget controls the resizing grid for its top-level window. This option is typically used in text widgets, where the information in the widget has a natural size (the size of a character) and it makes sense for the window's dimensions to be integral numbers of these units. These natural window sizes form a grid. If the setGrid option is set to true then the widget will communicate with the window manager so that when the user interactively resizes the top-level window that contains the widget, the dimensions of the window will be displayed to the user in grid units and the window size will be constrained to integral numbers of grid units. See the section **GRIDDED GEOMETRY MANAGEMENT** in the `wm` manual entry for more details.

**Command-Line Name:** –takefocus

**Database Name:** takeFocus

**Database Class: TakeFocus**

Determines whether the window accepts the focus during keyboard traversal (e.g., Tab and Shift–Tab). Before setting the focus to a window, the traversal scripts consult the value of the takeFocus option. A value of 0 means that the window should be skipped entirely during keyboard traversal. 1 means that the window should receive the input focus as long as it is viewable (it and all of its ancestors are mapped). An empty value for the option means that the traversal scripts make the decision about whether or not to focus on the window: the current algorithm is to skip the window if it is disabled, if it has no key bindings, or if it is not viewable. If the value has any other form, then the traversal scripts take the value, append the name of the window to it (with a separator space), and evaluate the resulting string as a Tcl script. The script must return 0, 1, or an empty string: a 0 or 1 value specifies whether the window will receive the input focus, and an empty string results in the default decision described above. Note: this interpretation of the option is defined entirely by the Tcl scripts that implement traversal: the widget implementations ignore the option entirely, so you can change its meaning if you redefine the keyboard traversal scripts.

**Command-Line Name:** –text

**Database Name:** text

**Database Class: Text**

Specifies a string to be displayed inside the widget. The way in which the string is displayed depends on the particular widget and may be determined by other options, such as anchor or justify.

**Command-Line Name:** –textvariable
Database Name: textVariable
Database Class: Variable
Specifies the name of a variable. The value of the variable is a text string to be displayed inside the widget; if the variable value changes then the widget will automatically update itself to reflect the new value. The way in which the string is displayed in the widget depends on the particular widget and may be determined by other options, such as anchor or justify.

Command–Line Name: –troughcolor
Database Name: troughColor
Database Class: Background
Specifies the color to use for the rectangular trough areas in widgets such as scrollbars and scales. This option is ignored for scrollbars on Windows (native widget doesn't recognize this option).

Command–Line Name: –underline
Database Name: underline
Database Class: Underline
Specifies the integer index of a character to underline in the widget. This option is used by the default bindings to implement keyboard traversal for menu buttons and menu entries. 0 corresponds to the first character of the text displayed in the widget, 1 to the next character, and so on.

Command–Line Name: –wraplength
Database Name: wrapLength
Database Class: WrapLength
For widgets that can perform word–wrapping, this option specifies the maximum line length. Lines that would exceed this length are wrapped onto the next line, so that no line is longer than the specified length. The value may be specified in any of the standard forms for screen distances. If this value is less than or equal to 0 then no wrapping is done: lines will break only at newline characters in the text.

Command–Line Name: –xscrollcommand
Database Name: xScrollCommand
Database Class: ScrollCommand
Specifies the prefix for a command used to communicate with horizontal scrollbars. When the view in the widget's window changes (or whenever anything else occurs that could change the display in a scrollbar, such as a change in the total size of the widget's contents), the widget will generate a Tcl command by concatenating the scroll command and two numbers. Each of the numbers is a fraction between 0 and 1, which indicates a position in the document. 0 indicates the beginning of the document, 1 indicates the end, .333 indicates a position one third the way through the document, and so on. The first fraction indicates the first information in the document that is visible in the window, and the second fraction indicates the information just after the last portion that is visible. The command is then passed to the Tcl interpreter for execution. Typically the xScrollCommand option consists of the path name of a scrollbar widget followed by ``set'', e.g. \`.x.scrollbar set'': this will cause the scrollbar to be updated whenever the view in the window changes. If this option is not specified, then no command will be executed.

Command–Line Name: –yscrollcommand
Database Name: yScrollCommand
Database Class: ScrollCommand
Specifies the prefix for a command used to communicate with vertical scrollbars. This option is treated in the same way as the xScrollCommand option, except that it is used for vertical scrollbars.
and is provided by widgets that support vertical scrolling. See the description of `xScrollCommand` for details on how this option is used.

SEE ALSO

colors, cursors, font

KEYWORDS

class, name, standard option, switch

Copyright © 1990−1994 The Regents of the University of California.
Copyright © 1994−1996 Sun Microsystems, Inc.
Copyright © 1995−1997 Roger E. Critchlow Jr.
NAME

pack – Geometry manager that packs around edges of cavity

SYNOPSIS

pack option arg ?arg...?

DESCRIPTION

pack slave ?slave...? ?options?
pack configure slave ?slave...? ?options?
  -after other
  -anchor anchor
  -before other
  -expand boolean
  -fill style
    none
    x
    y
    both
  -in other
  -ipadx amount
  -ipady amount
  -padx amount
  -pady amount
  -side side
pack forget slave ?slave...?
pack info slave
pack propagate master ?boolean?
pack slaves master

THE PACKER ALGORITHM

EXPANSION

GEOMETRY PROPAGATION

RESTRICTIONS ON MASTER WINDOWS

PACKING ORDER

EXAMPLE

SEE ALSO

KEYWORDS

NAME

pack – Geometry manager that packs around edges of cavity
SYNOPSIS

pack option arg ?arg ...?

DESCRIPTION

The pack command is used to communicate with the packer, a geometry manager that arranges the children of a parent by packing them in order around the edges of the parent. The pack command can have any of several forms, depending on the option argument:

pack slave ?slave ...? ?options?

If the first argument to pack is a window name (any value starting with `.'), then the command is processed in the same way as pack configure.

pack configure slave ?slave ...? ?options?

The arguments consist of the names of one or more slave windows followed by pairs of arguments that specify how to manage the slaves. See ``THE PACKER ALGORITHM'' below for details on how the options are used by the packer. The following options are supported:

−after other
  Other must the name of another window. Use its master as the master for the slaves, and insert the slaves just after other in the packing order.

−anchor anchor
  Anchor must be a valid anchor position such as n or sw; it specifies where to position each slave in its parcel. Defaults to center.

−before other
  Other must the name of another window. Use its master as the master for the slaves, and insert the slaves just before other in the packing order.

−expand boolean
  Specifies whether the slaves should be expanded to consume extra space in their master. Boolean may have any proper boolean value, such as 1 or no. Defaults to 0.

−fill style
  If a slave's parcel is larger than its requested dimensions, this option may be used to stretch the slave. Style must have one of the following values:

  none
    Give the slave its requested dimensions plus any internal padding requested with −ipadx or −ipady. This is the default.

  x
    Stretch the slave horizontally to fill the entire width of its parcel (except leave external padding as specified by −padx).

  y
    Stretch the slave vertically to fill the entire height of its parcel (except leave external padding as specified by −pady).

  both
    Stretch the slave both horizontally and vertically.

THE PACKER ALGORITHM

The packer uses the following algorithm to determine how to position the slaves:

1. For each slave, determine its requested dimensions.
2. If the slave's parcel is larger than its requested dimensions, use the −fill option to stretch the slave.
3. For each slave, calculate the amount of padding it needs to fit in its parcel.
4. Use the −anchor and −after options to position each slave in its parcel.
5. Use the −before and −after options to insert the slaves just after or before other in the packing order.
6. Use the −expand option to expand the slaves to consume extra space in their master.
7. Use the −fill option to stretch the slaves horizontally or vertically.

The packer uses this algorithm to determine how to position the slaves in the parent window.
−in other
Insert the slave(s) at the end of the packing order for the master window given by other.

−ipadx amount
Amount specifies how much horizontal internal padding to leave on each side of the slave(s). Amount must be a valid screen distance, such as 2 or .5c. It defaults to 0.

−ipady amount
Amount specifies how much vertical internal padding to leave on each side of the slave(s). Amount defaults to 0.

−padx amount
Amount specifies how much horizontal external padding to leave on each side of the slave(s). Amount may be a list of two values to specify padding for left and right separately. Amount defaults to 0.

−pady amount
Amount specifies how much vertical external padding to leave on each side of the slave(s). Amount may be a list of two values to specify padding for top and bottom separately. Amount defaults to 0.

−side side
Specifies which side of the master the slave(s) will be packed against. Must be left, right, top, or bottom. Defaults to top.

If no −in, −after or −before option is specified then each of the slaves will be inserted at the end of the packing list for its parent unless it is already managed by the packer (in which case it will be left where it is). If one of these options is specified then all the slaves will be inserted at the specified point. If any of the slaves are already managed by the geometry manager then any unspecified options for them retain their previous values rather than receiving default values.

pack forget slave ?slave ...?
Removes each of the slaves from the packing order for its master and unmaps their windows. The slaves will no longer be managed by the packer.

pack info slave
Returns a list whose elements are the current configuration state of the slave given by slave in the same option−value form that might be specified to pack configure. The first two elements of the list are "−in master" where master is the slave's master.

pack propagate master ?boolean?
If boolean has a true boolean value such as 1 or on then propagation is enabled for master, which must be a window name (see "GEOMETRY PROPAGATION" below). If boolean has a false boolean value then propagation is disabled for master. In either of these cases an empty string is returned. If boolean is omitted then the command returns 0 or 1 to indicate whether propagation is currently enabled for master. Propagation is enabled by default.

pack slaves master
Returns a list of all of the slaves in the packing order for master. The order of the slaves in the list is the same as their order in the packing order. If master has no slaves then an empty string is returned.

THE PACKER ALGORITHM

For each master the packer maintains an ordered list of slaves called the packing list. The −in, −after, and −before configuration options are used to specify the master for each slave and the slave's position in the
packing list. If none of these options is given for a slave then the slave is added to the end of the packing list for its parent.

The packer arranges the slaves for a master by scanning the packing list in order. At the time it processes each slave, a rectangular area within the master is still unallocated. This area is called the cavity; for the first slave it is the entire area of the master.

For each slave the packer carries out the following steps:

1. The packer allocates a rectangular parcel for the slave along the side of the cavity given by the slave's −side option. If the side is top or bottom then the width of the parcel is the width of the cavity and its height is the requested height of the slave plus the −ipady and −pady options. For the left or right side the height of the parcel is the height of the cavity and the width is the requested width of the slave plus the −ipadx and −padx options. The parcel may be enlarged further because of the −expand option (see "EXPANSION" below).

2. The packer chooses the dimensions of the slave. The width will normally be the slave's requested width plus twice its −ipadx option and the height will normally be the slave's requested height plus twice its −ipady option. However, if the −fill option is x or both then the width of the slave is expanded to fill the width of the parcel, minus twice the −padx option. If the −fill option is y or both then the height of the slave is expanded to fill the width of the parcel, minus twice the −pady option.

3. The packer positions the slave over its parcel. If the slave is smaller than the parcel then the −anchor option determines where in the parcel the slave will be placed. If −padx or −pady is non-zero, then the given amount of external padding will always be left between the slave and the edges of the parcel.

Once a given slave has been packed, the area of its parcel is subtracted from the cavity, leaving a smaller rectangular cavity for the next slave. If a slave doesn't use all of its parcel, the unused space in the parcel will not be used by subsequent slaves. If the cavity should become too small to meet the needs of a slave then the slave will be given whatever space is left in the cavity. If the cavity shrinks to zero size, then all remaining slaves on the packing list will be unmapped from the screen until the master window becomes large enough to hold them again.

EXPANSION

If a master window is so large that there will be extra space left over after all of its slaves have been packed, then the extra space is distributed uniformly among all of the slaves for which the −expand option is set. Extra horizontal space is distributed among the expandable slaves whose −side is left or right, and extra vertical space is distributed among the expandable slaves whose −side is top or bottom.
GEOMETRY PROPAGATION

The packer normally computes how large a master must be to just exactly meet the needs of its slaves, and it sets the requested width and height of the master to these dimensions. This causes geometry information to propagate up through a window hierarchy to a top−level window so that the entire sub−tree sizes itself to fit the needs of the leaf windows. However, the pack propagate command may be used to turn off propagation for one or more masters. If propagation is disabled then the packer will not set the requested width and height of the packer. This may be useful if, for example, you wish for a master window to have a fixed size that you specify.

RESTRICTIONS ON MASTER WINDOWS

The master for each slave must either be the slave's parent (the default) or a descendant of the slave's parent. This restriction is necessary to guarantee that the slave can be placed over any part of its master that is visible without danger of the slave being clipped by its parent.

PACKING ORDER

If the master for a slave is not its parent then you must make sure that the slave is higher in the stacking order than the master. Otherwise the master will obscure the slave and it will appear as if the slave hasn't been packed correctly. The easiest way to make sure the slave is higher than the master is to create the master window first: the most recently created window will be highest in the stacking order. Or, you can use the raise and lower commands to change the stacking order of either the master or the slave.

EXAMPLE

```tcl
# Make the widgets
label .t -text "This widget is at the top" -bg red
label .b -text "This widget is at the bottom" -bg green
label .l -text "Left\nHand\nSide"
label .r -text "Right\nHand\nSide"
text .mid
.mid insert end "This layout is like Java's BorderLayout"
# Lay then out
pack .t -side top -fill x
pack .b -side bottom -fill x
pack .l -side left -fill y
pack .r -side right -fill y
pack .mid -expand 1 -fill both
```

SEE ALSO

grid, place
NAME

panedwindow – Create and manipulate panedwindow widgets

SYNOPSIS

panedwindow pathName ?options?

STANDARD OPTIONS

−background or −bg, background, Background
−borderwidth or −bd, borderWidth, BorderWidth
−cursor, cursor, Cursor
−height
−orient, orient, Orient
−relief, relief, Relief
−width

WIDGET–SPECIFIC OPTIONS

−handlepad, handlePad, HandlePad
−handlesize, handleSize, HandleSize
−opaqueresize, opaqueResize, OpaqueResize
−sashcursor, sashCursor, SashCursor
−sashpad, sashPad, SashPad
−sashrelief, sashRelief, SashRelief
−sashwidth, sashWidth, SashWidth
−showhandle, showHandle, ShowHandle

DESCRIPTION

WIDGET COMMAND

pathName add window ?window ...? ?option value ...?

pathName cget option

pathName configure ?option? ?value option value ...?

pathName forget window ?window ...?

pathName identify x y

pathName proxy ?args?

pathName proxy coord

pathName proxy forget

pathName proxy place x y

pathName sash ?args?

pathName sash coord index

pathName sash dragto index x y

pathName sash mark index x y

pathName sash place index x y

pathName paneget window option

pathName paneconfigure window ?option? ?value option value ...?

−after window

−before window
NAME

panedwindow – Create and manipulate panedwindow widgets

SYNOPSIS

panedwindow pathName ?options?

STANDARD OPTIONS

-foreground or -fg, foreground, Background
-borderwidth or -bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-height
-orient, orient, Orient
-relief, relief, Relief
-width

WIDGET–SPECIFIC OPTIONS

Command–Line Name: -handlepad
Database Name: handlePad
Database Class: HandlePad
When sash handles are drawn, specifies the distance from the top or left end of the sash (depending on the orientation of the widget) at which to draw the handle. May be any value accepted by Tk_GetPixels.

Command–Line Name: -handlesize
Database Name: handleSize
Database Class: HandleSize
Specifies the side length of a sash handle. Handles are always drawn as squares. May be any value accepted by Tk_GetPixels.

Command–Line Name: -opaqueresize
Database Name: opaqueResize
Database Class: OpaqueResize
Specifies whether panes should be resized as a sash is moved (true), or if resizing should be deferred until the sash is placed (false).

Command Line Name: −sashcursor
Database Name: sashCursor
Database Class: SashCursor
Mouse cursor to use when over a sash. If null, sb_h_double_arrow will be used for horizontal panedwindows, and sb_v_double_arrow will be used for vertical panedwindows.

Command Line Name: −sashpad
Database Name: sashPad
Database Class: SashPad
Specifies the amount of padding to leave of each side of a sash. May be any value accepted by Tk_GetPixels.

Command Line Name: −sashrelief
Database Name: sashRelief
Database Class: SashRelief
Relief to use when drawing a sash. May be any of the standard Tk relief values.

Command Line Name: −sashwidth
Database Name: sashWidth
Database Class: SashWidth
Specifies the width of each sash. May be any value accepted by Tk_GetPixels.

Command Line Name: −showhandle
Database Name: showHandle
Database Class: ShowHandle
Specifies whether sash handles should be shown. May be any valid Tcl boolean value.

DESCRIPTION

The panedwindow command creates a new window (given by the pathName argument) and makes it into a panedwindow widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the panedwindow such as its default background color and relief. The panedwindow command returns the path name of the new window.

A panedwindow widget contains any number of panes, arranged horizontally or vertically, according to the value of the −orient option. Each pane contains one widget, and each pair of panes is separated by a moveable (via mouse movements) sash. Moving a sash causes the widgets on either side of the sash to be resized.

WIDGET COMMAND

The panedwindow command creates a new Tcl command whose name is the same as the path name of the panedwindow's window. This command may be used to invoke various operations on the widget. It has the following general form:
"pathName option ?arg arg ...?"

PathName is the name of the command, which is the same as the panedwindow widget's path name. Option
and the *args* determine the exact behavior of the command. The following commands are possible for panedwindow widgets:

**pathName add** window ?window ...? ?option value ...?
Add one or more windows to the panedwindow, each in a separate pane. The arguments consist of the names of one or more windows followed by pairs of arguments that specify how to manage the windows. *Option* may have any of the values accepted by the *configure* subcommand.

**pathName cget** option
Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *panedwindow* command.

**pathName configure** ?option? ?value option value ...?
Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see *Tk_ConfigureInfo* for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more option=value pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the *panedwindow* command.

**pathName forget** window ?window ...?
Remove the pane containing *window* from the panedwindow. All geometry management options for *window* will be forgotten.

**pathName identify** x y
Identify the panedwindow component underneath the point given by *x* and *y*, in window coordinates. If the point is over a sash or a sash handle, the result is a two element list containing the index of the sash or handle, and a word indicating whether it is over a sash or a handle, such as {0 sash} or {2 handle}. If the point is over any other part of the panedwindow, the result is an empty list.

**pathName proxy** ?args?
This command is used to query and change the position of the sash proxy, used for rubberband–style pane resizing. It can take any of the following forms:

**pathName proxy coord**
Return a list containing the x and y coordinates of the most recent proxy location.

**pathName proxy forget**
Remove the proxy from the display.

**pathName proxy place** x y
Place the proxy at the given *x* and *y* coordinates.

**pathName sash** ?args?
This command is used to query and change the position of sashes in the panedwindow. It can take any of the following forms:

**pathName sash coord**
Return the current x and y coordinate pair for the sash given by *index*. *Index* must be an integer between 0 and 1 less than the number of panes in the panedwindow. The coordinates given are those of the top left corner of the region containing the sash.

**pathName sash dragto** index x y
This command computes the difference between the given coordinates and the coordinates given to the last sash coord command for the given sash. It then moves that sash the computed difference. The return value is the empty string.

\texttt{pathName sash mark index \texttt{x y}}

Records \texttt{x} and \texttt{y} for the sash given by \texttt{index}; used in conjunction with later dragto commands to move the sash.

\texttt{pathName sash place index \texttt{x y}}

Place the sash given by \texttt{index} at the given coordinates.

\texttt{pathName paneconfigure window \texttt{?option? ?value option value ...?}}

Query or modify the management options for \texttt{window}. \texttt{Option} may be any value allowed by the paneconfigure subcommand.

\texttt{pathName paneget window option}

Query a management option for \texttt{window}. \texttt{Option} may be any value allowed by the paneconfigure subcommand.

The following options are supported:

\texttt{−after window}

Insert the window after the window specified. \texttt{window} should be the name of a window already managed by \texttt{pathName}.

\texttt{−before window}

Insert the window before the window specified. \texttt{window} should be the name of a window already managed by \texttt{pathName}.

\texttt{−height size}

Specify a height for the window. The height will be the outer dimension of the window including its border, if any. If \texttt{size} is an empty string, or if \texttt{−height} is not specified, then the height requested internally by the window will be used initially; the height may later be adjusted by the movement of sashes in the panedwindow. \texttt{Size} may be any value accepted by \texttt{Tk\_GetPixels}.

\texttt{−minsize n}

Specifies that the size of the window cannot be made less than \texttt{n}. This constraint only affects the size of the widget in the paned dimension — the \texttt{x} dimension for horizontal panedwindows, the \texttt{y} dimension for vertical panedwindows. May be any value accepted by \texttt{Tk\_GetPixels}.

\texttt{−padx n}

Specifies a non-negative value indicating how much extra space to leave on each side of the window in the \texttt{X}−direction. The value may have any of the forms accepted by \texttt{Tk\_GetPixels}.

\texttt{−pady n}

Specifies a non-negative value indicating how much extra space to leave on each side of the window in the \texttt{Y}−direction. The value may have any of the forms accepted by \texttt{Tk\_GetPixels}.

\texttt{−sticky style}

If a window's pane is larger than the requested dimensions of the window, this option may be
used to position (or stretch) the window within its pane. *Style* is a string that contains zero or more of the characters *n*, *s*, *e* or *w*. The string can optionally contain spaces or commas, but they are ignored. Each letter refers to a side (north, south, east, or west) that the window will "stick" to. If both *n* and *s* (or *e* and *w*) are specified, the window will be stretched to fill the entire height (or width) of its cavity.

−*width* *size*

Specify a width for the window. The width will be the outer dimension of the window including its border, if any. If *size* is an empty string, or if −*width* is not specified, then the width requested internally by the window will be used initially; the width may later be adjusted by the movement of sashes in the panedwindow. *Size* may be any value accepted by *Tk_GetPixels*.

*pathName* panes

Returns an ordered list of the widgets managed by *pathName*.

RESIZING PANES

A pane is resized by grabbing the sash (or sash handle if present) and dragging with the mouse. This is accomplished via mouse motion bindings on the widget. When a sash is moved, the sizes of the panes on each side of the sash, and thus the widgets in those panes, are adjusted.

When a pane is resized from outside (e.g. it is packed to expand and fill, and the containing toplevel is resized), space is added to the final (rightmost or bottommost) pane in the window.

KEYWORDS

*panedwindow*, *widget*, *geometry management*
NAME

photo – Full–color images

SYNOPSIS

image create photo ?name? ?options?

DESCRIPTION

CREATING PHOTOS

imageName blank
imageName cget option
imageName configure ?option? ?value option value ...?
imageName copy sourceImage ?option value(s) ...?

imageName data ?option value(s) ...?

imageName get x y
imageName put data ?option value(s) ...?

imageName read filename ?option value(s) ...?

imageName redither
imageName transparency subcommand ?arg arg ...?

imageName transparency get x y
**NAME**

`photo` – Full-color images

**SYNOPSIS**

```
image create photo ?name? ?options?
```

**DESCRIPTION**

A photo is an image whose pixels can display any color or be transparent. A photo image is stored internally in full color (32 bits per pixel), and is displayed using dithering if necessary. Image data for a photo image can be obtained from a file or a string, or it can be supplied from C code through a procedural interface. At present, only GIF and PPM/PGM formats are supported, but an interface exists to allow additional image file formats to be added easily. A photo image is transparent in regions where no image data has been supplied or where it has been set transparent by the `transparency set` subcommand.

**CREATING PHOTOS**

Like all images, photos are created using the `image create` command. Photos support the following `options`:

- `−data string`
  Specifies the contents of the image as a string. The string can contain base64 encoded data or binary data. The format of the string must be one of those for which there is an image file format handler that will accept string data. If both the `−data` and `−file` options are specified, the `−file` option takes precedence.

- `−format format−name`
  Specifies the name of the file format for the data specified with the `−data` or `−file` option.

- `−file name`
  `name` gives the name of a file that is to be read to supply data for the photo image. The file format must be one of those for which there is an image file format handler that can read data.
--gamma value
Specifies that the colors allocated for displaying this image in a window should be corrected for a non-linear display with the specified gamma exponent value. (The intensity produced by most CRT displays is a power function of the input value, to a good approximation; gamma is the exponent and is typically around 2). The value specified must be greater than zero. The default value is one (no correction). In general, values greater than one will make the image lighter, and values less than one will make it darker.

--height number
Specifies the height of the image, in pixels. This option is useful primarily in situations where the user wishes to build up the contents of the image piece by piece. A value of zero (the default) allows the image to expand or shrink vertically to fit the data stored in it.

--palette palette–spec
Specifies the resolution of the color cube to be allocated for displaying this image, and thus the number of colors used from the colormaps of the windows where it is displayed. The palette–spec string may be either a single decimal number, specifying the number of shades of gray to use, or three decimal numbers separated by slashes (/), specifying the number of shades of red, green and blue to use, respectively. If the first form (a single number) is used, the image will be displayed in monochrome (i.e., grayscale).

--width number
Specifies the width of the image, in pixels. This option is useful primarily in situations where the user wishes to build up the contents of the image piece by piece. A value of zero (the default) allows the image to expand or shrink horizontally to fit the data stored in it.

IMAGE COMMAND

When a photo image is created, Tk also creates a new command whose name is the same as the image. This command may be used to invoke various operations on the image. It has the following general form:

imageName option ?arg arg ...?

Option and the args determine the exact behavior of the command.

Those options that write data to the image generally expand the size of the image, if necessary, to accommodate the data written to the image, unless the user has specified non-zero values for the --width and/or --height configuration options, in which case the width and/or height, respectively, of the image will not be changed.

The following commands are possible for photo images:

imageName blank
Blank the image; that is, set the entire image to have no data, so it will be displayed as transparent, and the background of whatever window it is displayed in will show through.

imageName cget option
Returns the current value of the configuration option given by option. Option may have any of the values accepted by the image create photo command.
**imageName configure ?option? ?value option value ...?**

Query or modify the configuration options for the image. If no `option` is specified, returns a list describing all of the available options for `imageName` (see `Tk ConfigureInfo` for information on the format of this list). If `option` is specified with no `value`, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no `option` is specified). If one or more `option−value` pairs are specified, then the command modifies the given option(s) to have the given value(s); in this case the command returns an empty string. `Option` may have any of the values accepted by the `image create photo` command.

**imageName copy sourceImage ?option value(s) ...?**

Copies a region from the image called `sourceImage` (which must be a photo image) to the image called `imageName`, possibly with pixel zooming and/or subsampling. If no options are specified, this command copies the whole of `sourceImage` into `imageName`, starting at coordinates (0,0) in `imageName`. The following options may be specified:

- **−from x1 y1 x2 y2**
  Specifies a rectangular sub−region of the source image to be copied. `(x1,y1)` and `(x2,y2)` specify diagonally opposite corners of the rectangle. If `x2` and `y2` are not specified, the default value is the bottom−right corner of the source image. The pixels copied will include the left and top edges of the specified rectangle but not the bottom or right edges. If the `−from` option is not given, the default is the whole source image.

- **−to x1 y1 x2 y2**
  Specifies a rectangular sub−region of the destination image to be affected. `(x1,y1)` and `(x2,y2)` specify diagonally opposite corners of the rectangle. If `x2` and `y2` are not specified, the default value is `(x1,y1)` plus the size of the source region (after subsampling and zooming, if specified). If `x2` and `y2` are specified, the source region will be replicated if necessary to fill the destination region in a tiled fashion.

- **−shrink**
  Specifies that the size of the destination image should be reduced, if necessary, so that the region being copied into is at the bottom−right corner of the image. This option will not affect the width or height of the image if the user has specified a non−zero value for the `−width` or `−height` configuration option, respectively.

- **−zoom x y**
  Specifies that the source region should be magnified by a factor of `x` in the X direction and `y` in the Y direction. If `y` is not given, the default value is the same as `x`. With this option, each pixel in the source image will be expanded into a block of `x x y` pixels in the destination image, all the same color. `x` and `y` must be greater than 0.

- **−subsample x y**
  Specifies that the source image should be reduced in size by using only every `x`th pixel in the X direction and `y`th pixel in the Y direction. Negative values will cause the image to be flipped about the Y or X axes, respectively. If `y` is not given, the default value is the same as `x`.

- **−compositingrule rule**
  Specifies how transparent pixels in the source image are combined with the destination image. When a compositing rule of `overlay` is set, the old contents of the destination image are visible, as if the source image were printed on a piece of transparent film and placed over
the top of the destination. When a compositing rule of set is set, the old contents of the destination image are discarded and the source image is used as–is. The default compositing rule is overlay.

`imageName data?option value(s) ...?`

Returns image data in the form of a string. The following options may be specified:

- **background color**
  If the color is specified, the data will not contain any transparency information. In all transparent pixels the color will be replaced by the specified color.

- **format format–name**
  Specifies the name of the image file format handler to be used. Specifically, this subcommand searches for the first handler whose name matches an initial substring of format–name and which has the capability to read this image data. If this option is not given, this subcommand uses the first handler that has the capability to read the image data.

- **from x1 y1 x2 y2**
  Specifies a rectangular region of imageName to be returned. If only x1 and y1 are specified, the region extends from (x1,y1) to the bottom–right corner of imageName. If all four coordinates are given, they specify diagonally opposite corners of the rectangular region, including x1,y1 and excluding x2,y2. The default, if this option is not given, is the whole image.

- **grayscale**
  If this options is specified, the data will not contain color information. All pixel data will be transformed into grayscale.

`imageName get x y`

Returns the color of the pixel at coordinates (x,y) in the image as a list of three integers between 0 and 255, representing the red, green and blue components respectively.

`imageName put data?option value(s) ...?`

Sets pixels in imageName to the data specified in data. This command first searches the list of image file format handlers for a handler that can interpret the data in data, and then reads the image encoded within into imageName (the destination image). If data does not match any known format, an attempt to interpret it as a (top–to–bottom) list of scan–lines is made, with each scan–line being a (left–to–right) list of pixel colors (see `Tk_GetColor` for a description of valid colors.) Every scan–line must be of the same length. Note that when data is a single color name, you are instructing Tk to fill a rectangular region with that color. The following options may be specified:

- **format format–name**
  Specifies the format of the image data in data. Specifically, only image file format handlers whose names begin with format–name will be used while searching for an image data format handler to read the data.

- **to x1 y1 ?x2 y2?**
  Specifies the coordinates of the top–left corner (x1,y1) of the region of imageName into which data from filename are to be read. The default is (0,0). If x2,y2 is given and data is not large enough to cover the rectangle specified by this option, the image data extracted will be tiled so it covers the entire destination rectangle. Note that if data specifies a single color value, then a region extending to the bottom–right corner represented by (x2,y2) will be filled.
with that color.

**imageName read filename ?option value(s) ...?**

Reads image data from the file named `filename` into the image. This command first searches the list of image file format handlers for a handler that can interpret the data in `filename`, and then reads the image in `filename` into `imageName` (the destination image). The following options may be specified:

---

*format format-name*

Specifies the format of the image data in `filename`. Specifically, only image file format handlers whose names begin with `format-name` will be used while searching for an image data format handler to read the data.

*from x1 y1 x2 y2*

Specifies a rectangular sub-region of the image file data to be copied to the destination image. If only `x1` and `y1` are specified, the region extends from `(x1, y1)` to the bottom-right corner of the image file. If all four coordinates are specified, they specify diagonally opposite corners or the region. The default, if this option is not specified, is the whole of the image in the image file.

*shrink*

If this option, the size of `imageName` will be reduced, if necessary, so that the region into which the image file data are read is at the bottom-right corner of the `imageName`. This option will not affect the width or height of the image if the user has specified a non-zero value for the *−width* or *−height* configuration option, respectively.

*to x y*

Specifies the coordinates of the top-left corner of the region of `imageName` into which data from `filename` are to be read. The default is `(0,0).

**imageName redither**

The dithering algorithm used in displaying photo images propagates quantization errors from one pixel to its neighbors. If the image data for `imageName` is supplied in pieces, the dithered image may not be exactly correct. Normally the difference is not noticeable, but if it is a problem, this command can be used to recalculate the dithered image in each window where the image is displayed.

**imageName transparency subcommand ?arg arg ...?**

Allows examination and manipulation of the transparency information in the photo image. Several subcommands are available:

*imageName transparency get x y*

Returns a boolean indicating if the pixel at `(x,y)` is transparent.

*imageName transparency set x y boolean*

Makes the pixel at `(x,y)` transparent if `boolean` is true, and makes that pixel opaque otherwise.

**imageName write filename ?option value(s) ...?**

Writes image data from `imageName` to a file named `filename`. The following options may be specified:

*background color*

If the color is specified, the data will not contain any transparency information. In all transparent pixels the color will be replaced by the specified color.

*format format-name*
Specifies the name of the image file format handler to be used to write the data to the file. Specifically, this subcommand searches for the first handler whose name matches an initial substring of `format-name` and which has the capability to write an image file. If this option is not given, this subcommand uses the first handler that has the capability to write an image file.

`-from x1 y1 x2 y2`

Specifies a rectangular region of `imageName` to be written to the image file. If only `x1` and `y1` are specified, the region extends from `(x1,y1)` to the bottom–right corner of `imageName`. If all four coordinates are given, they specify diagonally opposite corners of the rectangular region. The default, if this option is not given, is the whole image.

`-grayscale`

If this options is specified, the data will not contain color information. All pixel data will be transformed into grayscale.

**IMAGE FORMATS**

The photo image code is structured to allow handlers for additional image file formats to be added easily. The photo image code maintains a list of these handlers. Handlers are added to the list by registering them with a call to `Tk_CreatePhotoImageFormat`. The standard Tk distribution comes with handlers for PPM/PGM and GIF formats, which are automatically registered on initialization.

When reading an image file or processing string data specified with the `-data` configuration option, the photo image code invokes each handler in turn until one is found that claims to be able to read the data in the file or string. Usually this will find the correct handler, but if it doesn't, the user may give a format name with the `-format` option to specify which handler to use. In fact the photo image code will try those handlers whose names begin with the string specified for the `-format` option (the comparison is case–insensitive). For example, if the user specifies `-format gif`, then a handler named GIF87 or GIF89 may be invoked, but a handler named JPEG may not (assuming that such handlers had been registered).

When writing image data to a file, the processing of the `-format` option is slightly different: the string value given for the `-format` option must begin with the complete name of the requested handler, and may contain additional information following that, which the handler can use, for example, to specify which variant to use of the formats supported by the handler. Note that not all image handlers may support writing transparency data to a file, even where the target image format does.

**COLOR ALLOCATION**

When a photo image is displayed in a window, the photo image code allocates colors to use to display the image and dithers the image, if necessary, to display a reasonable approximation to the image using the colors that are available. The colors are allocated as a color cube, that is, the number of colors allocated is the product of the number of shades of red, green and blue.

Normally, the number of colors allocated is chosen based on the depth of the window. For example, in an 8–bit PseudoColor window, the photo image code will attempt to allocate seven shades of red, seven shades of green and four shades of blue, for a total of 198 colors. In a 1–bit StaticGray (monochrome) window, it will...
allocate two colors, black and white. In a 24−bit DirectColor or TrueColor window, it will allocate 256 shades each of red, green and blue. Fortunately, because of the way that pixel values can be combined in DirectColor and TrueColor windows, this only requires 256 colors to be allocated. If not all of the colors can be allocated, the photo image code reduces the number of shades of each primary color and tries again.

The user can exercise some control over the number of colors that a photo image uses with the −palette configuration option. If this option is used, it specifies the maximum number of shades of each primary color to try to allocate. It can also be used to force the image to be displayed in shades of gray, even on a color display, by giving a single number rather than three numbers separated by slashes.

CREDITS

The photo image type was designed and implemented by Paul Mackerras, based on his earlier photo widget and some suggestions from John Ousterhout.

EXAMPLE

Load an image from a file and tile it to the size of a window, which is useful for producing a tiled background:

# These lines should be called once
image create photo untiled -file "theFile.ppm"
image create photo tiled

# These lines should be called whenever .someWidget changes
# size; a <Configure> binding is useful here
set width [winfo width .someWidget]
set height [winfo height .someWidget]
tiled copy untiled -to 0 0 $width $height -shrink

SEE ALSO

image

KEYWORDS

photo, image, color

Copyright ' 1994 The Australian National University
Copyright ' 1994-1997 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
place

NAME

place – Geometry manager for fixed or rubber–sheet placement

SYNOPSIS

place option arg ?arg ...?

DESCRIPTION

place window option value ?option value ...?
place configure window ?option ?value option value ...?

-anchor where
-bordermode mode
-height size
-in master
-relheight size
-relwidth size
-relx location
-rely location
-width size
-x location
-y location

place forget window
place info window
place slaves window

FINE POINTS
EXAMPLE
SEE ALSO
KEYWORDS

NAME

place – Geometry manager for fixed or rubber–sheet placement

SYNOPSIS

place option arg ?arg ...?

DESCRIPTION

The placer is a geometry manager for Tk. It provides simple fixed placement of windows, where you specify the exact size and location of one window, called the slave, within another window, called the master. The placer also provides rubber–sheet placement, where you specify the size and location of the slave in terms of the dimensions of the master, so that the slave changes size and location in response to changes in the size of
the master. Lastly, the placer allows you to mix these styles of placement so that, for example, the slave has a fixed width and height but is centered inside the master.

`place window option value ?option value ...?`

Arrange for the placer to manage the geometry of a slave whose pathName is `window`. The remaining arguments consist of one or more `option−value` pairs that specify the way in which `window`'s geometry is managed. `Option` may have any of the values accepted by the `place configure` command.

`place configure window ?option? ?value option value ...?`

Query or modify the geometry options of the slave given by `window`. If no `option` is specified, this command returns a list describing the available options (see `Tk_ConfigureInfo` for information on the format of this list). If `option` is specified with no `value`, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no `option` is specified). If one or more `option−value` pairs are specified, then the command modifies the given option(s) to have the given value(s); in this case the command returns an empty string. The following `option−value` pairs are supported:

`−anchor where`

`Where` specifies which point of `window` is to be positioned at the (x,y) location selected by the `−x`, `−y`, `−relx`, and `−rely` options. The anchor point is in terms of the outer area of `window` including its border, if any. Thus if `where` is `se` then the lower−right corner of `window`'s border will appear at the given (x,y) location in the master. The anchor position defaults to `nw`.

`−bordermode mode`

`Mode` determines the degree to which borders within the master are used in determining the placement of the slave. The default and most common value is `inside`. In this case the placer considers the area of the master to be the innermost area of the master, inside any border: an option of `−x 0` corresponds to an x−coordinate just inside the border and an option of `−relwidth 1.0` means `window` will fill the area inside the master's border. If `mode` is `outside` then the placer considers the area of the master to include its border; this mode is typically used when placing `window` outside its master, as with the options `−x 0 −y 0 −anchor ne`. Lastly, `mode` may be specified as `ignore`, in which case borders are ignored: the area of the master is considered to be its official X area, which includes any internal border but no external border. A bordermode of `ignore` is probably not very useful.

`−height size`

`Size` specifies the height for `window` in screen units (i.e. any of the forms accepted by `Tk_GetPixels`). The height will be the outer dimension of `window` including its border, if any. If `size` is an empty string, or if no `−height` or `−relheight` option is specified, then the height requested internally by the window will be used.

`−in master`

`Master` specifies the path name of the window relative to which `window` is to be placed. `Master` must either be `window`'s parent or a descendant of `window`'s parent. In addition, `master` and `window` must both be descendants of the same top−level window. These restrictions are necessary to guarantee that `window` is visible whenever `master` is visible. If this option isn't specified then the master defaults to `window`'s parent.

`−relheight size`
Size specifies the height for \textit{window}. In this case the height is specified as a floating−point number relative to the height of the master: 0.5 means \textit{window} will be half as high as the master, 1.0 means \textit{window} will have the same height as the master, and so on. If both \texttt{-height} and \texttt{-relheight} are specified for a slave, their values are summed. For example, \texttt{-relheight 1.0 -height 2} makes the slave 2 pixels shorter than the master.

\texttt{-relwidth size}

\textit{Size} specifies the width for \textit{window}. In this case the width is specified as a floating−point number relative to the width of the master: 0.5 means \textit{window} will be half as wide as the master, 1.0 means \textit{window} will have the same width as the master, and so on. If both \texttt{-width} and \texttt{-relwidth} are specified for a slave, their values are summed. For example, \texttt{-relwidth 1.0 -width 5} makes the slave 5 pixels wider than the master.

\texttt{-relx location}

\textit{Location} specifies the x−coordinate within the master window of the anchor point for \textit{window}. In this case the location is specified in a relative fashion as a floating−point number: 0.0 corresponds to the left edge of the master and 1.0 corresponds to the right edge of the master. \textit{Location} need not be in the range 0.0−1.0. If both \texttt{-x} and \texttt{-relx} are specified for a slave then their values are summed. For example, \texttt{-relx 0.5 -x 2} positions the left edge of the slave 2 pixels to the left of the center of its master.

\texttt{-rely location}

\textit{Location} specifies the y−coordinate within the master window of the anchor point for \textit{window}. In this case the value is specified in a relative fashion as a floating−point number: 0.0 corresponds to the top edge of the master and 1.0 corresponds to the bottom edge of the master. \textit{Location} need not be in the range 0.0−1.0. If both \texttt{-y} and \texttt{-rely} are specified for a slave then their values are summed. For example, \texttt{-rely 0.5 -y 2} positions the top edge of the slave 3 pixels below the center of its master.

\texttt{-width size}

\textit{Size} specifies the width for \textit{window} in screen units (i.e. any of the forms accepted by \texttt{Tk GetPixels}). The width will be the outer width of \textit{window} including its border, if any. If \texttt{size} is an empty string, or if no \texttt{-width} or \texttt{-relwidth} option is specified, then the width requested internally by the window will be used.

\texttt{-x location}

\textit{Location} specifies the x−coordinate within the master window of the anchor point for \textit{window}. The location is specified in screen units (i.e. any of the forms accepted by \texttt{Tk GetPixels}) and need not lie within the bounds of the master window.

\texttt{-y location}

\textit{Location} specifies the y−coordinate within the master window of the anchor point for \textit{window}. The location is specified in screen units (i.e. any of the forms accepted by \texttt{Tk GetPixels}) and need not lie within the bounds of the master window.

If the same value is specified separately with two different options, such as \texttt{-x} and \texttt{-relx}, then the most recent option is used and the older one is ignored.

\texttt{place forget window}

Causes the placer to stop managing the geometry of \textit{window}. As a side effect of this command \textit{window} will be unmapped so that it doesn't appear on the screen. If \textit{window} isn't currently managed by the placer then the command has no effect. This command returns an empty string.

\texttt{place info window}
Returns a list giving the current configuration of window. The list consists of option−value pairs in exactly the same form as might be specified to the place configure command.

place slaves window
Returns a list of all the slave windows for which window is the master. If there are no slaves for window then an empty string is returned.

If the configuration of a window has been retrieved with place info, that configuration can be restored later by first using place forget to erase any existing information for the window and then invoking place configure with the saved information.

FINE POINTS

It is not necessary for the master window to be the parent of the slave window. This feature is useful in at least two situations. First, for complex window layouts it means you can create a hierarchy of subwindows whose only purpose is to assist in the layout of the parent. The ``real children" of the parent (i.e. the windows that are significant for the application's user interface) can be children of the parent yet be placed inside the windows of the geometry−management hierarchy. This means that the path names of the ``real children" don't reflect the geometry−management hierarchy and users can specify options for the real children without being aware of the structure of the geometry−management hierarchy.

A second reason for having a master different than the slave's parent is to tie two siblings together. For example, the placer can be used to force a window always to be positioned centered just below one of its siblings by specifying the configuration

```
−in sibling −relx 0.5 −rely 1.0 −anchor n −bordermode outside
```

Whenever the sibling is repositioned in the future, the slave will be repositioned as well.

Unlike many other geometry managers (such as the packer) the placer does not make any attempt to manipulate the geometry of the master windows or the parents of slave windows (i.e. it doesn't set their requested sizes). To control the sizes of these windows, make them windows like frames and canvases that provide configuration options for this purpose.

EXAMPLE

Make the label occupy the middle bit of the toplevel, no matter how it is resized:

```
label .l −text "In the\nMiddle!" −bg black −fg white
place .l −relwidth .3 −relx .35 −relheight .3 −rely .35
```

SEE ALSO

grid, pack
NAME
radiobutton – Create and manipulate radiobutton widgets

SYNOPSIS
radiobutton pathName ?options?

STANDARD OPTIONS
- activebackground, activeBackground, Foreground
- activeforeground, activeForeground, Background
- anchor, anchor, Anchor
- background or -bg, background, Background
- bitmap, bitmap, Bitmap
- borderwidth or -bd, borderWidth, BorderWidth
- compound, compound, Compound
- cursor, cursor, Cursor
- disabledforeground, disabledForeground, DisabledForeground
- font, font, Font
- foreground or -fg, foreground, Foreground
- highlightbackground, highlightBackground, HighlightBackground
- highlightcolor, highlightColor, HighlightColor
- highlightthickness, highlightThickness, HighlightThickness
- image, image, Image
- justify, justify, Justify
- padx, padX, Pad
- pady, padY, Pad
- relief, relief, Relief
- takefocus, takeFocus, TakeFocus
- text, text, Text
- textvariable, textVariable, Variable
- underline, underline, Underline
- wraplength, wrapLength, WrapLength

WIDGET–SPECIFIC OPTIONS
- command, command, Command
- height, height, Height
- indicatoron, indicatorOn, IndicatorOn
- selectcolor, selectColor, Background
- offrelief, offRelief, OffRelief
- overrelief, overRelief, OverRelief
- selectimage, selectImage, SelectImage
- state, state, State
- value, value, Value
- variable, variable, Variable
- width, width, Width
DESCRIPTION

WIDGET COMMAND

pathName cget option
pathName configure ?option? ?value option value ...?
pathName deselect
pathName flash
pathName invoke
pathName select

BINDINGS

SEE ALSO

KEYWORDS

NAME

radiobutton – Create and manipulate radiobutton widgets

SYNOPSIS

radiobutton pathName ?options?

STANDARD OPTIONS

-activebackground, activeBackground, Foreground
-activeforeground, activeForeground, Background
-anchor, anchor, Anchor
-background or -bg, background, Background
-bitmap, bitmap, Bitmap
-borderwidth or -bd, borderWidth, BorderWidth
-compound, compound, Compound
-cursor, cursor, Cursor
-disabledforeground, disabledForeground, DisabledForeground
-font, font, Font
-foreground or -fg, foreground, Foreground
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-image, image, Image
-justify, justify, Justify
-padX, padX, Pad
-padY, padY, Pad
-relief, relief, Relief
-takefocus, takeFocus, TakeFocus
-text, text, Text
-textvariable, textVariable, Variable
-underline, underline, Underline
WIDGET–SPECIFIC OPTIONS

Command–Line Name: −command
Database Name: command
Database Class: Command

Specifies a Tcl command to associate with the button. This command is typically invoked when mouse button 1 is released over the button window. The button's global variable (−variable option) will be updated before the command is invoked.

Command–Line Name: −height
Database Name: height
Database Class: Height

Specifies a desired height for the button. If an image or bitmap is being displayed in the button then the value is in screen units (i.e. any of the forms acceptable to Tk_GetPixels); for text it is in lines of text. If this option isn't specified, the button's desired height is computed from the size of the image or bitmap or text being displayed in it.

Command–Line Name: −indicatoron
Database Name: indicatorOn
Database Class: IndicatorOn

Specifies whether or not the indicator should be drawn. Must be a proper boolean value. If false, the relief option is ignored and the widget's relief is always sunken if the widget is selected and raised otherwise.

Command–Line Name: −selectcolor
Database Name: selectColor
Database Class: Background

Specifies a background color to use when the button is selected. If indicatorOn is true then the color applies to the indicator. Under Windows, this color is used as the background for the indicator regardless of the select state. If indicatorOn is false, this color is used as the background for the entire widget, in place of background or activeBackground, whenever the widget is selected. If specified as an empty string then no special color is used for displaying when the widget is selected.

Command–Line Name: −offrelief
Database Name: offRelief
Database Class: OffRelief

Specifies the relief for the checkbutton when the indicator is not drawn and the checkbutton is off. The default value is "raised". By setting this option to "flat" and setting −indicatoron to false and −overrelief to raised, the effect is achieved of having a flat button that raises on mouse-over and which is depressed when activated. This is the behavior typically exhibited by the Align–Left, Align–Right, and Center radiobuttons on the toolbar of a word–processor, for example.

Command–Line Name: −overrelief
Database Name: overRelief
Database Class: OverRelief

Specifies an alternative relief for the radiobutton, to be used when the mouse cursor is over the widget. This option can be used to make toolbar buttons, by configuring −relief flat −overrelief raised. If the value of this option is the empty string, then no alternative relief is used when the mouse
cursor is over the radiobutton. The empty string is the default value.

Command–Line Name: −selectimage
Database Name: selectImage
Database Class: SelectImage

Specifies an image to display (in place of the image option) when the radiobutton is selected. This option is ignored unless the image option has been specified.

Command–Line Name: −state
Database Name: state
Database Class: State

Specifies one of three states for the radiobutton: normal, active, or disabled. In normal state the radiobutton is displayed using the foreground and background options. The active state is typically used when the pointer is over the radiobutton. In active state the radiobutton is displayed using the activeForeground and activeBackground options. Disabled state means that the radiobutton should be insensitive: the default bindings will refuse to activate the widget and will ignore mouse button presses. In this state the disabledForeground and background options determine how the radiobutton is displayed.

Command–Line Name: −value
Database Name: value
Database Class: Value

Specifies value to store in the button's associated variable whenever this button is selected.

Command–Line Name: −variable
Database Name: variable
Database Class: Variable

Specifies name of global variable to set whenever this button is selected. Changes in this variable also cause the button to select or deselect itself. Defaults to the value selectedButton.

Command–Line Name: −width
Database Name: width
Database Class: Width

Specifies a desired width for the button. If an image or bitmap is being displayed in the button, the value is in screen units (i.e. any of the forms acceptable to Tk_GetPixels); for text it is in characters. If this option isn't specified, the button's desired width is computed from the size of the image or bitmap or text being displayed in it.

DESCRIPTION

The radiobutton command creates a new window (given by the pathName argument) and makes it into a radiobutton widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the radiobutton such as its colors, font, text, and initial relief. The radiobutton command returns its pathName argument. At the time this command is invoked, there must not exist a window named pathName, but pathName's parent must exist.

A radiobutton is a widget that displays a textual string, bitmap or image and a diamond or circle called an indicator. If text is displayed, it must all be in a single font, but it can occupy multiple lines on the screen (if it contains newlines or if wrapping occurs because of the wrapLength option) and one of the characters may optionally be underlined using the underline option. A radiobutton has all of the behavior of a simple button:
it can display itself in either of three different ways, according to the state option; it can be made to appear raised, sunken, or flat; it can be made to flash; and it invokes a Tcl command whenever mouse button 1 is clicked over the check button.

In addition, radiobuttons can be selected. If a radiobutton is selected, the indicator is normally drawn with a selected appearance, and a Tcl variable associated with the radiobutton is set to a particular value (normally 1). Under Unix, the indicator is drawn with a sunken relief and a special color. Under Windows, the indicator is drawn with a round mark inside. If the radiobutton is not selected, then the indicator is drawn with a deselected appearance, and the associated variable is set to a different value (typically 0). Under Unix, the indicator is drawn with a raised relief and no special color. Under Windows, the indicator is drawn without a round mark inside. Typically, several radiobuttons share a single variable and the value of the variable indicates which radiobutton is to be selected. When a radiobutton is selected it sets the value of the variable to indicate that fact; each radiobutton also monitors the value of the variable and automatically selects and deselects itself when the variable's value changes. By default the variable selectedButton is used; its contents give the name of the button that is selected, or the empty string if no button associated with that variable is selected. The name of the variable for a radiobutton, plus the variable to be stored into it, may be modified with options on the command line or in the option database. Configuration options may also be used to modify the way the indicator is displayed (or whether it is displayed at all). By default a radiobutton is configured to select itself on button clicks.

**WIDGET COMMAND**

The radiobutton command creates a new Tcl command whose name is pathName. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

Option and the args determine the exact behavior of the command. The following commands are possible for radiobutton widgets:

```
pathName cget option
```

Returns the current value of the configuration option given by option. Option may have any of the values accepted by the radiobutton command.

```
pathName configure ?option? ?value option value ...?
```

Query or modify the configuration options of the widget. If no option is specified, returns a list describing all of the available options for pathName (see Tk_ConfigureInfo for information on the format of this list). If option is specified with no value, the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no option is specified). If one or more option-value pairs are specified, the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. Option may have any of the values accepted by the radiobutton command.

```
pathName deselect
```

Deselects the radiobutton and sets the associated variable to an empty string. If this radiobutton was not currently selected, the command has no effect.

```
pathName flash
```

WIDGET COMMAND
Flashes the radiobutton. This is accomplished by redisplaying the radiobutton several times, alternating between active and normal colors. At the end of the flash the radiobutton is left in the same normal/active state as when the command was invoked. This command is ignored if the radiobutton's state is disabled.

`pathName invoke`

Does just what would have happened if the user invoked the radiobutton with the mouse: selects the button and invokes its associated Tcl command, if there is one. The return value is the return value from the Tcl command, or an empty string if there is no command associated with the radiobutton. This command is ignored if the radiobutton's state is disabled.

`pathName select`

Selects the radiobutton and sets the associated variable to the value corresponding to this widget.

BINDINGS

Tk automatically creates class bindings for radiobuttons that give them the following default behavior:

[1]
On Unix systems, a radiobutton activates whenever the mouse passes over it and deactivates whenever the mouse leaves the radiobutton. On Mac and Windows systems, when mouse button 1 is pressed over a radiobutton, the button activates whenever the mouse pointer is inside the button, and deactivates whenever the mouse pointer leaves the button.

[2]
When mouse button 1 is pressed over a radiobutton it is invoked (it becomes selected and the command associated with the button is invoked, if there is one).

[3]
When a radiobutton has the input focus, the space key causes the radiobutton to be invoked.

If the radiobutton's state is disabled then none of the above actions occur: the radiobutton is completely non-responsive.

The behavior of radiobuttons can be changed by defining new bindings for individual widgets or by redefining the class bindings.

SEE ALSO

checkbutton, labelframe, listbox, options, scale

KEYWORDS

radiobutton, widget
raise

NAME
raise – Change a window's position in the stacking order

SYNOPSIS
raise window ?aboveThis?

DESCRIPTION
If the aboveThis argument is omitted then the command raises window so that it is above all of its siblings in the stacking order (it will not be obscured by any siblings and will obscure any siblings that overlap it). If aboveThis is specified then it must be the path name of a window that is either a sibling of window or the descendant of a sibling of window. In this case the raise command will insert window into the stacking order just above aboveThis (or the ancestor of aboveThis that is a sibling of window); this could end up either raising or lowering window.

EXAMPLE
Make a button appear to be in a sibling frame that was created after it. This is is often necessary when building GUIs in the style where you create your activity widgets first before laying them out on the display:

button .b -text "Hi there!"
pack [frame .f -background blue]
pack [label .f.l1 -text "This is above"]
pack .b -in .f
pack [label .f.l2 -text "This is below"]
raise .b

SEE ALSO
lower

KEYWORDS
obscure, raise, stacking order

Copyright ' 1990 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

scale – Create and manipulate scale widgets

SYNOPSIS

    scale pathName ?options?

STANDARD OPTIONS

    -activebackground, activeBackground, Foreground
    -background or -bg, background, Background
    -borderwidth or -bd, borderWidth, BorderWidth
    -cursor, cursor, Cursor
    -font, font, Font
    -foreground or -fg, foreground, Foreground
    -highlightbackground, highlightBackground, HighlightBackground
    -highlightcolor, highlightColor, HighlightColor
    -highlightthickness, highlightThickness, HighlightThickness
    -orient, orient, Orient
    -relief, relief, Relief
    -repeatdelay, repeatDelay, RepeatDelay
    -repeatinterval, repeatInterval, RepeatInterval
    -takefocus, takeFocus, TakeFocus
    -troughcolor, troughColor, Background

WIDGET–SPECIFIC OPTIONS

    -bigincrement, bigIncrement, BigIncrement
    -command, command, Command
    -digits, digits, Digits
    -from, from, From
    -label, label, Label
    -length, length, Length
    -resolution, resolution, Resolution
    -showvalue, showValue, ShowValue
    -sliderlength, sliderLength, SliderLength
    -sliderrelief, sliderRelief, SliderRelief
    -state, state, State
    -tickinterval, tickInterval, TickInterval
    -to, to, To
    -variable, variable, Variable
    -width, width, Width

DESCRIPTION

WIDGET COMMAND

    pathName cget option
    pathName configure ?option? ?value option value ...?
    pathName coords ?value?
pathName get ?x y?  
pathName identify x y  
pathName set value

BINDINGS
KEYWORDS

NAME

scale – Create and manipulate scale widgets

SYNOPSIS

scale pathName ?options?

STANDARD OPTIONS

-activebackground, activeBackground, Foreground
-background or –bg, background, Background
-borderwidth or –bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-font, font, Font
-foreground or –fg, foreground, Foreground
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-orient, orient, Orient
-relief, relief, Relief
-repeatdelay, repeatDelay, RepeatDelay
-repeatinterval, repeatInterval, RepeatInterval
-takefocus, takeFocus, TakeFocus
-troughcolor, troughColor, Background

WIDGET–SPECIFIC OPTIONS

Command–Line Name: –bigincrement
Database Name: bigIncrement
Database Class: BigIncrement

Some interactions with the scale cause its value to change by ``large'' increments; this option specifies
the size of the large increments. If specified as 0, the large increments default to 1/10 the range of the
scale.

Command–Line Name: –command
Database Name: command
Database Class: Command

Specifies the prefix of a Tcl command to invoke whenever the scale's value is changed via a widget
command. The actual command consists of this option followed by a space and a real number indicating the new value of the scale.

**Command-Line Name:** −digits
**Database Name:** digits
**Database Class:** Digits

An integer specifying how many significant digits should be retained when converting the value of the scale to a string. If the number is less than or equal to zero, then the scale picks the smallest value that guarantees that every possible slider position prints as a different string.

**Command-Line Name:** −from
**Database Name:** from
**Database Class:** From

A real value corresponding to the left or top end of the scale.

**Command-Line Name:** −label
**Database Name:** label
**Database Class:** Label

A string to display as a label for the scale. For vertical scales the label is displayed just to the right of the top end of the scale. For horizontal scales the label is displayed just above the left end of the scale. If the option is specified as an empty string, no label is displayed.

**Command-Line Name:** −length
**Database Name:** length
**Database Class:** Length

Specifies the desired long dimension of the scale in screen units (i.e. any of the forms acceptable to Tk_GetPixels). For vertical scales this is the scale's height; for horizontal scales it is the scale's width.

**Command-Line Name:** −resolution
**Database Name:** resolution
**Database Class:** Resolution

A real value specifying the resolution for the scale. If this value is greater than zero then the scale's value will always be rounded to an even multiple of this value, as will tick marks and the endpoints of the scale. If the value is less than zero then no rounding occurs. Defaults to 1 (i.e., the value will be integral).

**Command-Line Name:** −showvalue
**Database Name:** showValue
**Database Class:** ShowValue

Specifies a boolean value indicating whether or not the current value of the scale is to be displayed.

**Command-Line Name:** −sliderlength
**Database Name:** sliderLength
**Database Class:** SliderLength

Specifies the size of the slider, measured in screen units along the slider's long dimension. The value may be specified in any of the forms acceptable to Tk_GetPixels.

**Command-Line Name:** −sliderrelief
**Database Name:** sliderRelief
**Database Class:** SliderRelief

Specifies the relief to use when drawing the slider, such as raised or sunken.

**Command-Line Name:** −state
**Database Name:** state
**Database Class:** State
**Database Class: State**

Specifies one of three states for the scale: normal, active, or disabled. If the scale is disabled then the value may not be changed and the scale won't activate. If the scale is active, the slider is displayed using the color specified by the activeBackground option.

**Command–Line Name:** −tickinterval
**Database Name:** tickInterval
**Database Class:** TickInterval

Must be a real value. Determines the spacing between numerical tick marks displayed below or to the left of the slider. If 0, no tick marks will be displayed.

**Command–Line Name:** −to
**Database Name:** to
**Database Class:** To

Specifies a real value corresponding to the right or bottom end of the scale. This value may be either less than or greater than the from option.

**Command–Line Name:** −variable
**Database Name:** variable
**Database Class:** Variable

Specifies the name of a global variable to link to the scale. Whenever the value of the variable changes, the scale will update to reflect this value. Whenever the scale is manipulated interactively, the variable will be modified to reflect the scale's new value.

**Command–Line Name:** −width
**Database Name:** width
**Database Class:** Width

Specifies the desired narrow dimension of the trough in screen units (i.e. any of the forms acceptable to Tk_GetPixels). For vertical scales this is the trough's width; for horizontal scales this is the trough's height.

**DESCRIPTION**

The scale command creates a new window (given by the pathName argument) and makes it into a scale widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the scale such as its colors, orientation, and relief. The scale command returns its pathName argument. At the time this command is invoked, there must not exist a window named pathName, but pathName's parent must exist.

A scale is a widget that displays a rectangular trough and a small slider. The trough corresponds to a range of real values (determined by the from, to, and resolution options), and the position of the slider selects a particular real value. The slider's position (and hence the scale's value) may be adjusted with the mouse or keyboard as described in the BINDINGS section below. Whenever the scale's value is changed, a Tcl command is invoked (using the command option) to notify other interested widgets of the change. In addition, the value of the scale can be linked to a Tcl variable (using the variable option), so that changes in either are reflected in the other.

Three annotations may be displayed in a scale widget: a label appearing at the top right of the widget (top left for horizontal scales), a number displayed just to the left of the slider (just above the slider for horizontal
scales), and a collection of numerical tick marks just to the left of the current value (just below the trough for horizontal scales). Each of these three annotations may be enabled or disabled using the configuration options.

**WIDGET COMMAND**

The **scale** command creates a new Tcl command whose name is *pathName*. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

*Option* and the *args* determine the exact behavior of the command. The following commands are possible for scale widgets:

**pathName cget option**

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **scale** command.

**pathName configure ?option? ?value option value ...?**

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **scale** command.

**pathName coords ?value?**

Returns a list whose elements are the x and y coordinates of the point along the centerline of the trough that corresponds to *value*. If *value* is omitted then the scale's current value is used.

**pathName get ?x y?**

If *x* and *y* are omitted, returns the current value of the scale. If *x* and *y* are specified, they give pixel coordinates within the widget; the command returns the scale value corresponding to the given pixel. Only one of *x* or *y* is used: for horizontal scales *y* is ignored, and for vertical scales *x* is ignored.

**pathName identify x y**

Returns a string indicating what part of the scale lies under the coordinates given by *x* and *y*. A return value of **slider** means that the point is over the slider; **trough1** means that the point is over the portion of the slider above or to the left of the slider; and **trough2** means that the point is over the portion of the slider below or to the right of the slider. If the point isn't over one of these elements, an empty string is returned.

**pathName set value**

This command is invoked to change the current value of the scale, and hence the position at which the slider is displayed. *Value* gives the new value for the scale. The command has no effect if the scale is disabled.
BINDINGS

Tk automatically creates class bindings for scales that give them the following default behavior. Where the behavior is different for vertical and horizontal scales, the horizontal behavior is described in parentheses.

[1] If button 1 is pressed in the trough, the scale's value will be incremented or decremented by the value of the resolution option so that the slider moves in the direction of the cursor. If the button is held down, the action auto−repeats.

[2] If button 1 is pressed over the slider, the slider can be dragged with the mouse.

[3] If button 1 is pressed in the trough with the Control key down, the slider moves all the way to the end of its range, in the direction towards the mouse cursor.

[4] If button 2 is pressed, the scale's value is set to the mouse position. If the mouse is dragged with button 2 down, the scale's value changes with the drag.

[5] The Up and Left keys move the slider up (left) by the value of the resolution option.

[6] The Down and Right keys move the slider down (right) by the value of the resolution option.

[7] Control−Up and Control−Left move the slider up (left) by the value of the bigIncrement option.

[8] Control−Down and Control−Right move the slider down (right) by the value of the bigIncrement option.

[9] Home moves the slider to the top (left) end of its range.

[10] End moves the slider to the bottom (right) end of its range.

If the scale is disabled using the state option then none of the above bindings have any effect.

The behavior of scales can be changed by defining new bindings for individual widgets or by redefining the class bindings.

KEYWORDS

scale, slider, trough, widget
scrollbar

NAME

scrollbar – Create and manipulate scrollbar widgets

SYNOPSIS

scrollbar pathName ?options?

STANDARD OPTIONS

−activebackground, activeBackground, Foreground
−background or −bg, background, Background
−borderwidth or −bd, borderWidth, BorderWidth
−cursor, cursor, Cursor
−highlightbackground, highlightBackground, HighlightBackgroundColor
−highlightcolor, highlightColor, HighlightColor
−highlightthickness, highlightThickness, HighlightThickness
−jump, jump, Jump
−orient, orient, Orient
−relief, relief, Relief
−repeatdelay, repeatDelay, RepeatDelay
−repeatinterval, repeatInterval, RepeatInterval
−takefocus, takeFocus, TakeFocus
−troughcolor, troughColor, Background

WIDGET–SPECIFIC OPTIONS

−activerelief, activeRelief, ActiveRelief
−command, command, Command
−elementborderwidth, elementBorderWidth, BorderWidth
−width, width, Width

DESCRIPTION

ELEMENTS

arrow1
trough1
slider
trough2
arrow2

WIDGET COMMAND

pathName activate ?element?
pathName cget option
pathName configure ?option? ?value option value ...?
pathName delta deltaX deltaY
pathName fraction x y
pathName get
pathName identify x y
pathName set first last

SCROLLING COMMANDS
prefix moveto fraction
prefix scroll number units
prefix scroll number pages

OLD COMMAND SYNTAX
pathName set totalUnits windowUnits firstUnit lastUnit
prefix unit

BINDINGS
EXAMPLE
KEYWORDS

NAME
scrollbar – Create and manipulate scrollbar widgets

SYNOPSIS

scrollbar pathName ?options?

STANDARD OPTIONS

-activatebackground, activeBackground, Foreground
-background or -bg, background, Background
-borderwidth or -bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-jump, jump, Jump
-orient, orient, Orient
-relief, relief, Relief
-repeatdelay, repeatDelay, RepeatDelay
-repeatinterval, repeatInterval, RepeatInterval
-takefocus, takeFocus, TakeFocus
-troughcolor, troughColor, Background

WIDGET−SPECIFIC OPTIONS

Command−Line Name: −activefocus
Database Name: activeRelief
Database Class: ActiveRelief

Specifies the relief to use when displaying the element that is active, if any. Elements other than the active element are always displayed with a raised relief.

Command−Line Name: −command
Database Name: command
**Database Class: Command**  
Specifies the prefix of a Tcl command to invoke to change the view in the widget associated with the scrollbar. When a user requests a view change by manipulating the scrollbar, a Tcl command is invoked. The actual command consists of this option followed by additional information as described later. This option almost always has a value such as `.t xview` or `.t yview`, consisting of the name of a widget and either `xview` (if the scrollbar is for horizontal scrolling) or `yview` (for vertical scrolling). All scrollable widgets have `xview` and `yview` commands that take exactly the additional arguments appended by the scrollbar as described in SCROLLING COMMANDS below.

**Command–Line Name:** `-elementborderwidth`  
**Database Name:** `elementBorderWidth`  
**Database Class:** `BorderWidth`  
Specifies the width of borders drawn around the internal elements of the scrollbar (the two arrows and the slider). The value may have any of the forms acceptable to `Tk_GetPixels`. If this value is less than zero, the value of the `borderWidth` option is used in its place.

**Command–Line Name:** `-width`  
**Database Name:** `width`  
**Database Class:** `Width`  
Specifies the desired narrow dimension of the scrollbar window, not including 3-D border, if any. For vertical scrollbars this will be the width and for horizontal scrollbars this will be the height. The value may have any of the forms acceptable to `Tk_GetPixels`.

**DESCRIPTION**

The `scrollbar` command creates a new window (given by the `pathName` argument) and makes it into a scrollbar widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the scrollbar such as its colors, orientation, and relief. The `scrollbar` command returns its `pathName` argument. At the time this command is invoked, there must not exist a window named `pathName`, but `pathName`'s parent must exist.

A scrollbar is a widget that displays two arrows, one at each end of the scrollbar, and a slider in the middle portion of the scrollbar. It provides information about what is visible in an associated window that displays a document of some sort (such as a file being edited or a drawing). The position and size of the slider indicate which portion of the document is visible in the associated window. For example, if the slider in a vertical scrollbar covers the top third of the area between the two arrows, it means that the associated window displays the top third of its document.

Scrollbars can be used to adjust the view in the associated window by clicking or dragging with the mouse. See the **BINDINGS** section below for details.

**ELEMENTS**

A scrollbar displays five elements, which are referred to in the widget commands for the scrollbar:

**arrow1**  
The top or left arrow in the scrollbar.
The region between the slider and **arrow1**.

The rectangle that indicates what is visible in the associated widget.

The region between the slider and **arrow2**.

The bottom or right arrow in the scrollbar.

**WIDGET COMMAND**

The **scrollbar** command creates a new Tcl command whose name is *pathName*. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

*Option* and the *args* determine the exact behavior of the command. The following commands are possible for scrollbar widgets:

```
pathName activate ?element?
```

Marks the element indicated by *element* as active, which causes it to be displayed as specified by the *activeBackground* and *activeRelief* options. The only element values understood by this command are **arrow1**, **slider**, or **arrow2**. If any other value is specified then no element of the scrollbar will be active. If *element* is not specified, the command returns the name of the element that is currently active, or an empty string if no element is active.

```
pathName cget option
```

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **scrollbar** command.

```
pathName configure ?option? ?value option value ...?
```

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **scrollbar** command.

```
pathName delta deltaX deltaY
```

Returns a real number indicating the fractional change in the scrollbar setting that corresponds to a given change in slider position. For example, if the scrollbar is horizontal, the result indicates how much the scrollbar setting must change to move the slider *deltaX* pixels to the right (*deltaY* is ignored in this case). If the scrollbar is vertical, the result indicates how much the scrollbar setting must change to move the slider *deltaY* pixels down. The arguments and the result may be zero or negative.

```
pathName fraction x y
```

Returns a real number between 0 and 1 indicating where the point given by *x* and *y* lies in the trough area of the scrollbar. The value 0 corresponds to the top or left of the trough, the value 1 corresponds
to the bottom or right, 0.5 corresponds to the middle, and so on. \(x\) and \(y\) must be pixel coordinates relative to the scrollbar widget. If \(x\) and \(y\) refer to a point outside the trough, the closest point in the trough is used.

\textit{pathName get}

Returns the scrollbar settings in the form of a list whose elements are the arguments to the most recent \texttt{set} widget command.

\textit{pathName identify \(x\) \(y\)}

Returns the name of the element under the point given by \(x\) and \(y\) (such as \texttt{arrow1}), or an empty string if the point does not lie in any element of the scrollbar. \(X\) and \(y\) must be pixel coordinates relative to the scrollbar widget.

\textit{pathName set \(first\) \(last\)}

This command is invoked by the scrollbar's associated widget to tell the scrollbar about the current view in the widget. The command takes two arguments, each of which is a real fraction between 0 and 1. The fractions describe the range of the document that is visible in the associated widget. For example, if \(first\) is 0.2 and \(last\) is 0.4, it means that the first part of the document visible in the window is 20\% of the way through the document, and the last visible part is 40\% of the way through.

### SCROLLING COMMANDS

When the user interacts with the scrollbar, for example by dragging the slider, the scrollbar notifies the associated widget that it must change its view. The scrollbar makes the notification by evaluating a Tcl command generated from the scrollbar's \texttt{−command} option. The command may take any of the following forms. In each case, \textit{prefix} is the contents of the \texttt{−command} option, which usually has a form like \texttt{.t yview}

**prefix moveto fraction**

\textit{Fraction} is a real number between 0 and 1. The widget should adjust its view so that the point given by \textit{fraction} appears at the beginning of the widget. If \textit{fraction} is 0 it refers to the beginning of the document. 1.0 refers to the end of the document, 0.333 refers to a point one-third of the way through the document, and so on.

**prefix scroll number units**

The widget should adjust its view by \textit{number} units. The units are defined in whatever way makes sense for the widget, such as characters or lines in a text widget. \textit{Number} is either 1, which means one unit should scroll off the top or left of the window, or −1, which means that one unit should scroll off the bottom or right of the window.

**prefix scroll number pages**

The widget should adjust its view by \textit{number} pages. It is up to the widget to define the meaning of a page; typically it is slightly less than what fits in the window, so that there is a slight overlap between the old and new views. \textit{Number} is either 1, which means the next page should become visible, or −1, which means that the previous page should become visible.

### OLD COMMAND SYNTAX

In versions of Tk before 4.0, the \texttt{set} and \texttt{get} widget commands used a different form. This form is still supported for backward compatibility, but it is deprecated. In the old command syntax, the \texttt{set} widget command has the following form:
pathName set totalUnits windowUnits firstUnit lastUnit
In this form the arguments are all integers. TotalUnits gives the total size of the object being displayed in the associated widget. The meaning of one unit depends on the associated widget; for example, in a text editor widget units might correspond to lines of text. WindowUnits indicates the total number of units that can fit in the associated window at one time. FirstUnit and lastUnit give the indices of the first and last units currently visible in the associated window (zero corresponds to the first unit of the object).

Under the old syntax the get widget command returns a list of four integers, consisting of the totalUnits, windowUnits, firstUnit, and lastUnit values from the last set widget command.

The commands generated by scrollbars also have a different form when the old syntax is being used:

prefix unit
   Unit is an integer that indicates what should appear at the top or left of the associated widget's window. It has the same meaning as the firstUnit and lastUnit arguments to the set widget command.

The most recent set widget command determines whether or not to use the old syntax. If it is given two real arguments then the new syntax will be used in the future, and if it is given four integer arguments then the old syntax will be used.

BINDINGS

Tk automatically creates class bindings for scrollbars that give them the following default behavior. If the behavior is different for vertical and horizontal scrollbars, the horizontal behavior is described in parentheses.

[1] Pressing button 1 over arrow1 causes the view in the associated widget to shift up (left) by one unit so that the document appears to move down (right) one unit. If the button is held down, the action auto–repeats.

[2] Pressing button 1 over trough1 causes the view in the associated widget to shift up (left) by one screenful so that the document appears to move down (right) one screenful. If the button is held down, the action auto–repeats.

[3] Pressing button 1 over the slider and dragging causes the view to drag with the slider. If the jump option is true, then the view doesn't drag along with the slider; it changes only when the mouse button is released.

[4] Pressing button 1 over trough2 causes the view in the associated widget to shift down (right) by one screenful so that the document appears to move up (left) one screenful. If the button is held down, the action auto–repeats.

[5] Pressing button 1 over arrow2 causes the view in the associated widget to shift down (right) by one unit so that the document appears to move up (left) one unit. If the button is held down, the action
auto-repeats.

[6] If button 2 is pressed over the trough or the slider, it sets the view to correspond to the mouse position; dragging the mouse with button 2 down causes the view to drag with the mouse. If button 2 is pressed over one of the arrows, it causes the same behavior as pressing button 1.

[7] If button 1 is pressed with the Control key down, then if the mouse is over arrow1 or trough1 the view changes to the very top (left) of the document; if the mouse is over arrow2 or trough2 the view changes to the very bottom (right) of the document; if the mouse is anywhere else then the button press has no effect.

[8] In vertical scrollbars the Up and Down keys have the same behavior as mouse clicks over arrow1 and arrow2, respectively. In horizontal scrollbars these keys have no effect.

[9] In vertical scrollbars Control–Up and Control–Down have the same behavior as mouse clicks over trough1 and trough2, respectively. In horizontal scrollbars these keys have no effect.

[10] In horizontal scrollbars the Up and Down keys have the same behavior as mouse clicks over arrow1 and arrow2, respectively. In vertical scrollbars these keys have no effect.

[11] In horizontal scrollbars Control–Up and Control–Down have the same behavior as mouse clicks over trough1 and trough2, respectively. In vertical scrollbars these keys have no effect.

[12] The Prior and Next keys have the same behavior as mouse clicks over trough1 and trough2, respectively.

[13] The Home key adjusts the view to the top (left edge) of the document.

[14] The End key adjusts the view to the bottom (right edge) of the document.

EXAMPLE

Create a window with a scrollable text widget:

toplevel .tl
text .tl.t -yscrollcommand {.tl.s set} scrollbar .tl.s -command {.tl.t yview} grid .tl.t .tl.s -sticky nsew grid columnconfigure .tl 0 -weight 1 grid rowconfigure .tl 0 -weight 1

KEYWORDS

scrollbar, widget

Copyright ' 1990-1994 The Regents of the University of California.
NAME

selection – Manipulate the X selection

SYNOPSIS

selection option ?arg arg ...?

DESCRIPTION

This command provides a Tcl interface to the X selection mechanism and implements the full selection functionality described in the X Inter–Client Communication Conventions Manual (ICCCM).

Note that for management of the CLIPBOARD selection (see below), the `clipboard` command may also be used.

The first argument to `selection` determines the format of the rest of the arguments and the behavior of the command. The following forms are currently supported:

`selection clear` ?–displayof window? ?–selection selection?

If `selection` exists anywhere on `window`'s display, clear it so that no window owns the selection anymore. `Selection` specifies the X selection that should be cleared, and should be an atom name such as PRIMARY or CLIPBOARD; see the Inter–Client Communication Conventions Manual for complete details. `Selection` defaults to PRIMARY and `window` defaults to ``````. Returns an empty string.

`selection get` ?–displayof window? ?–selection selection? ?–type type?

EXAMPLES

SEE ALSO

KEYWORDS

NAME

selection – Manipulate the X selection

SYNOPSIS

selection option ?arg arg ...?

DESCRIPTION

This command provides a Tcl interface to the X selection mechanism and implements the full selection functionality described in the X Inter–Client Communication Conventions Manual (ICCCM).

Note that for management of the CLIPBOARD selection (see below), the `clipboard` command may also be used.

The first argument to `selection` determines the format of the rest of the arguments and the behavior of the command. The following forms are currently supported:

`selection clear` ?–displayof window? ?–selection selection?

If `selection` exists anywhere on `window`'s display, clear it so that no window owns the selection anymore. `Selection` specifies the X selection that should be cleared, and should be an atom name such as PRIMARY or CLIPBOARD; see the Inter–Client Communication Conventions Manual for complete details. `Selection` defaults to PRIMARY and `window` defaults to ``````. Returns an empty string.

`selection get` ?–displayof window? ?–selection selection? ?–type type?
Retrieves the value of selection from window's display and returns it as a result. Selection defaults to PRIMARY and window defaults to ``.". Type specifies the form in which the selection is to be returned (the desired ``target'' for conversion, in ICCCM terminology), and should be an atom name such as STRING or FILE_NAME; see the Inter−Client Communication Conventions Manual for complete details. Type defaults to STRING. The selection owner may choose to return the selection in any of several different representation formats, such as STRING, ATOM, INTEGER, etc. (this format is different than the selection type; see the ICCCM for all the confusing details). If the selection is returned in a non−string format, such as INTEGER or ATOM, the selection command converts it to string format as a collection of fields separated by spaces: atoms are converted to their textual names, and anything else is converted to hexadecimal integers.


Creates a handler for selection requests, such that command will be executed whenever selection is owned by window and someone attempts to retrieve it in the form given by type (e.g. type is specified in the selection get command). Selection defaults to PRIMARY, type defaults to STRING, and format defaults to STRING. If command is an empty string then any existing handler for window, type, and selection is removed.

When selection is requested, window is the selection owner, and type is the requested type, command will be executed as a Tcl command with two additional numbers appended to it (with space separators). The two additional numbers are offset and maxChars: offset specifies a starting character position in the selection and maxChars gives the maximum number of characters to retrieve. The command should return a value consisting of at most maxChars of the selection, starting at position offset. For very large selections (larger than maxChars) the selection will be retrieved using several invocations of command with increasing offset values. If command returns a string whose length is less than maxChars, the return value is assumed to include all of the remainder of the selection; if the length of command's result is equal to maxChars then command will be invoked again, until it eventually returns a result shorter than maxChars. The value of maxChars will always be relatively large (thousands of characters).

If command returns an error then the selection retrieval is rejected just as if the selection didn't exist at all.

The format argument specifies the representation that should be used to transmit the selection to the requester (the second column of Table 2 of the ICCCM), and defaults to STRING. If format is STRING, the selection is transmitted as 8−bit ASCII characters (i.e. just in the form returned by command). If format is ATOM, then the return value from command is divided into fields separated by white space; each field is converted to its atom value, and the 32−bit atom value is transmitted instead of the atom name. For any other format, the return value from command is divided into fields separated by white space and each field is converted to a 32−bit integer; an array of integers is transmitted to the selection requester.

The format argument is needed only for compatibility with selection requesters that don't use Tk. If Tk is being used to retrieve the selection then the value is converted back to a string at the requesting end, so format is irrelevant.

**selection own ?−displayof window? ?−selection selection?**
selection own ?--command command? ?--selection selection? window

The first form of selection own returns the path name of the window in this application that owns selection on the display containing window, or an empty string if no window in this application owns the selection. Selection defaults to PRIMARY and window defaults to ``.".

The second form of selection own causes window to become the new owner of selection on window's display, returning an empty string as result. The existing owner, if any, is notified that it has lost the selection. If command is specified, it is a Tcl script to execute when some other window claims ownership of the selection away from window. Selection defaults to PRIMARY.

EXAMPLES

On X11 platforms, one of the standard selections available is the SECONDARY selection. Hardly anything uses it, but here is how to read it using Tk:

set selContents [selection get -selection SECONDARY]

Many different types of data may be available for a selection; the special type TARGETS allows you to get a list of available types:

foreach type [selection get -type TARGETS] {
    puts "Selection PRIMARY supports type $type"
}

To claim the selection, you must first set up a handler to supply the data for the selection. Then you have to claim the selection...

# Set up the data handler ready for incoming requests
set foo "This is a string with some data in it... blah blah"
selection handle -selection SECONDARY . getData
proc getData {offset maxChars} {
    puts "Retrieving selection starting at $offset"
    return [string range $::foo $offset [expr {$offset+$maxChars}]]
}

# Now we grab the selection itself
puts "Claiming selection"
selection own -command lost -selection SECONDARY .
proc lost {} {
    puts "Lost selection"
}

SEE ALSO

clipboard
KEYWORDS

clear, format, handler, ICCCM, own, selection, target, type

Copyright © 1990-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
Copyright © 1995-1997 Roger E. Critchlow Jr.
send

NAME
send – Execute a command in a different application

SYNOPSIS
send ?options? app cmd ?arg arg ...?

DESCRIPTION
−async
−displayof pathName

APPLICATION NAMES

DISABLING SENDS

SECURITY

EXAMPLE

KEYWORDS

NAME
send – Execute a command in a different application

SYNOPSIS
send ?options? app cmd ?arg arg ...?

DESCRIPTION
This command arranges for cmd (and args) to be executed in the application named by app. It returns the result or error from that command execution. App may be the name of any application whose main window is on the display containing the sender's main window; it need not be within the same process. If no arg arguments are present, then the command to be executed is contained entirely within the cmd argument. If one or more args are present, they are concatenated to form the command to be executed, just as for the eval command.

If the initial arguments of the command begin with ``−" they are treated as options. The following options are currently defined:

−async
Requests asynchronous invocation. In this case the send command will complete immediately without waiting for cmd to complete in the target application; no result will be available and errors in the sent command will be ignored. If the target application is in the same process as the sending application then the −async option is ignored.

−displayof pathName
Specifies that the target application's main window is on the display of the window given by `pathName`, instead of the display containing the application's main window.

\[\text{–} \]\[\text{–}\]

Serves no purpose except to terminate the list of options. This option is needed only if `app` could contain a leading "\-" character.

**APPLICATION NAMES**

The name of an application is set initially from the name of the program or script that created the application. You can query and change the name of an application with the `tk appname` command.

**DISABLING SENDS**

If the `send` command is removed from an application (e.g. with the command `rename send {}`) then the application will not respond to incoming send requests anymore, nor will it be able to issue outgoing requests. Communication can be reenabled by invoking the `tk appname` command.

**SECURITY**

The `send` command is potentially a serious security loophole. On Unix, any application that can connect to your X server can send scripts to your applications. These incoming scripts can use Tcl to read and write your files and invoke subprocesses under your name. Host-based access control such as that provided by `xhost` is particularly insecure, since it allows anyone with an account on particular hosts to connect to your server, and if disabled it allows anyone anywhere to connect to your server. In order to provide at least a small amount of security, Tk checks the access control being used by the server and rejects incoming sends unless (a) `xhost`−style access control is enabled (i.e. only certain hosts can establish connections) and (b) the list of enabled hosts is empty. This means that applications cannot connect to your server unless they use some other form of authorization such as that provide by `xauth`. Under Windows, `send` is currently disabled. Most of the functionality is provided by the `dde` command instead.

**EXAMPLE**

This script fragment can be used to make an application that only runs once on a particular display.

```tcl
if {[tk appname FoobarApp] ne "FoobarApp"} {
    send -async FoobarApp RemoteStart $argv
    exit
} # The command that will be called remotely, which raises
# the application main window and opens the requested files
proc RemoteStart args {
    raise .
    foreach filename $args {
        OpenFile $filename
    }
}
```
spinbox

NAME

spinbox – Create and manipulate spinbox widgets

SYNOPSIS

spinbox pathName ?options?

STANDARD OPTIONS

-activebackground, activeBackground, Foreground
-background or –bg, background, Background
-borderwidth or –bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-exportselection, exportSelection, ExportSelection
-font, font, Font
-foreground or –fg, foreground, Foreground
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-insertbackground, insertBackground, Foreground
-insertborderwidth, insertBorderWidth, BorderWidth
-insertofftime, insertOffTime, OffTime
-insertontime, insertOnTime, OnTime
-insertwidth, insertWidth, InsertWidth
-justify, justify, Justify
-relief, relief, Relief
-repeatdelay, repeatDelay, RepeatDelay
-repeatinterval, repeatInterval, RepeatInterval
-selectbackground, selectBackground, Foreground
-selectborderwidth, selectBorderWidth, BorderWidth
-selectforeground, selectForeground, Background
-takefocus, takeFocus, TakeFocus
-textvariable, textVariable, Variable
-xscrollcommand, xScrollCommand, ScrollCommand

WIDGET–SPECIFIC OPTIONS

-buttonbackground, buttonBackground, Background
-buttoncursor, buttonCursor, Cursor
-buttondownrelief, buttonDownRelief, Relief
-buttonuprelief, buttonUpRelief, Relief
-command, command, Command
-disabledbackground, disabledBackground, DisabledBackground
-disabledforeground, disabledForeground, DisabledForeground
-format, format, Format
-from, from, From
-invalidcommand or –invcmd, invalidCommand, InvalidCommand
- increment, increment, Increment
- readonlybackground, readonlyBackground, ReadonlyBackground
- state, state, State
- to, to, To
- validate, validate, Validate
- validatecommand or vcmd, validateCommand, ValidateCommand
- values, values, Values
- width, width, Width
- wrap, wrap, wrap

DESCRIPTION
VALIDATION
none
focus
focusin
focusout
key
all
%d
%i
%P
%s
%S
%v
%V
%W

WIDGET COMMAND
number
anchor
end
insert
sel.first
sel.last
@number
pathName bbox index
pathName cget option
pathName configure ?option? ?value option value ...?
pathName delete first ?last?
pathName get
pathName icursor index
pathName identify x y
pathName index index
pathName insert index string
pathName invoke element
pathName scan option args
    pathName scan mark x
pathName scan dragto x
pathName selection option arg
pathName selection adjust index
pathName selection clear
pathName selection element ?element?
pathName selection from index
pathName selection present
pathName selection range start end
pathName selection to index
pathName set ?string?
pathName validate
pathName xview args
pathName xview
pathName xview index
pathName xview moveto fraction
pathName xview scroll number what

DEFAULT BINDINGS
KEYWORDS

NAME

spinbox – Create and manipulate spinbox widgets

SYNOPSIS

spinbox pathName ?options?

STANDARD OPTIONS

-activebackground, activeBackground, Foreground
-background or –bg, background, Background
-borderwidth or –bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-exportselection, exportSelection, ExportSelection
-font, font, Font
-foreground or –fg, foreground, Foreground
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-insertbackground, insertBackground, Foreground
-insertborderwidth, insertBorderWidth, BorderWidth
-insertofftime, insertOffTime, OffTime
-insertonTime, insertOnTime, OnTime
-insertwidth, insertWidth, InsertWidth
-justify, justify, Justify
-relief, relief, Relief
-repeatdelay, repeatDelay, RepeatDelay
-repeatinterval, repeatInterval, RepeatInterval
-selectbackground, selectBackground, Foreground
-selectborderwidth, selectBorderWidth, BorderWidth
-selectforeground, selectForeground, Background
-takefocus, takeFocus, TakeFocus
-textvariable, textVariable, Variable
-xscrollcommand, xScrollCommand, ScrollCommand

WIDGET−SPECIFIC OPTIONS

Command−Line Name: −buttonbackground
Database Name: buttonBackground
Database Class: Background
   The background color to be used for the spin buttons.

Command−Line Name: −buttoncursor
Database Name: buttonCursor
Database Class: Cursor
   The cursor to be used when over the spin buttons. If this is empty (the default), a default cursor will be used.

Command−Line Name: −buttondownrelief
Database Name: buttonDownRelief
Database Class: Relief
   The relief to be used for the upper spin button.

Command−Line Name: −buttonuprelief
Database Name: buttonUpRelief
Database Class: Relief
   The relief to be used for the lower spin button.

Command−Line Name: −command
Database Name: command
Database Class: Command
   Specifies a Tcl command to invoke whenever a spinbutton is invoked. The command recognizes several percent substitutions: %W for the widget path, %s for the current value of the widget, and %d for the direction of the button pressed (up or down).

Command−Line Name: −disabledbackground
Database Name: disabledBackground
Database Class: DisabledBackground
   Specifies the background color to use when the spinbox is disabled. If this option is the empty string, the normal background color is used.

Command−Line Name: −disabledforeground
Database Name: disabledForeground
Database Class: DisabledForeground
   Specifies the foreground color to use when the spinbox is disabled. If this option is the empty string, the normal foreground color is used.
Command–Line Name: –format
Database Name: format
Database Class: Format

Specifies an alternate format to use when setting the string value when using the –from and –to range. This must be a format specifier of the form %<pad>.<pad>f, as it will format a floating-point number.

Command–Line Name: –from
Database Name: from
Database Class: From

A floating-point value corresponding to the lowest value for a spinbox, to be used in conjunction with –to and –increment. When all are specified correctly, the spinbox will use these values to control its contents. This value must be less than the –to option. If –values is specified, it supercedes this option.

Command–Line Name: –invalidcommand or –invcmd
Database Name: invalidCommand
Database Class: InvalidCommand

Specifies a script to eval when validateCommand returns 0. Setting it to an empty string disables this feature (the default). The best use of this option is to set it to bell. See Validation below for more information.

Command–Line Name: –increment
Database Name: increment
Database Class: Increment

A floating-point value specifying the increment. When used with –from and –to, the value in the widget will be adjusted by –increment when a spin button is pressed (up adds the value, down subtracts the value).

Command–Line Name: –readonlybackground
Database Name: readonlyBackground
Database Class: ReadonlyBackground

Specifies the background color to use when the spinbox is readonly. If this option is the empty string, the normal background color is used.

Command–Line Name: –state
Database Name: state
Database Class: State

Specifies one of three states for the spinbox: normal, disabled, or readonly. If the spinbox is readonly, then the value may not be changed using widget commands and no insertion cursor will be displayed, even if the input focus is in the widget; the contents of the widget may still be selected. If the spinbox is disabled, the value may not be changed, no insertion cursor will be displayed, the contents will not be selectable, and the spinbox may be displayed in a different color, depending on the values of the –disabledforeground and –disabledbackground options.

Command–Line Name: –to
Database Name: to
Database Class: To

A floating-point value corresponding to the highest value for the spinbox, to be used in conjunction with –from and –increment. When all are specified correctly, the spinbox will use these values to control its contents. This value must be greater than the –from option. If –values is specified, it supercedes this option.
Command—Line Name: −validate
Database Name: validate
Database Class: Validate

Specifies the mode in which validation should operate: none, focus, focusin, focusout, key, or all. It defaults to none. When you want validation, you must explicitly state which mode you wish to use. See Validation below for more.

Command—Line Name: −validatecommand or −vcmd
Database Name: validateCommand
Database Class: ValidateCommand

Specifies a script to evaluate when you want to validate the input in the widget. Setting it to an empty string disables this feature (the default). Validation occurs according to the value of −validate. This command must return a valid Tcl boolean value. If it returns 0 (or the valid Tcl boolean equivalent) then the value of the widget will not change and the invalidCommand will be evaluated if it is set. If it returns 1, then value will be changed. See Validation below for more information.

Command—Line Name: −values
Database Name: values
Database Class: Values

Must be a proper list value. If specified, the spinbox will use these values as to control its contents, starting with the first value. This option has precedence over the −from and −to range.

Command—Line Name: −width
Database Name: width
Database Class: Width

Specifies an integer value indicating the desired width of the spinbox window, in average-size characters of the widget's font. If the value is less than or equal to zero, the widget picks a size just large enough to hold its current text.

Command—Line Name: −wrap
Database Name: wrap
Database Class: wrap

Must be a proper boolean value. If on, the spinbox will wrap around the values of data in the widget.

DESCRIPTION

The spinbox command creates a new window (given by the pathName argument) and makes it into a spinbox widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the spinbox such as its colors, font, and relief. The spinbox command returns its pathName argument. At the time this command is invoked, there must not exist a window named pathName, but pathName's parent must exist.

A spinbox is an extended entry widget that allows he user to move, or spin, through a fixed set of ascending or descending values such as times or dates in addition to editing the value as in an entry. When first created, a spinbox's string is empty. A portion of the spinbox may be selected as described below. If a spinbox is exporting its selection (see the exportSelection option), then it will observe the standard protocols for handling the selection; spinbox selections are available as type STRING. Spinboxes also observe the standard Tk rules for dealing with the input focus. When a spinbox has the input focus it displays an insertion cursor to indicate where new characters will be inserted.
Spinboxes are capable of displaying strings that are too long to fit entirely within the widget’s window. In this case, only a portion of the string will be displayed; commands described below may be used to change the view in the window. Spinboxes use the standard `xScrollCommand` mechanism for interacting with scrollbars (see the description of the `xScrollCommand` option for details). They also support scanning, as described below.

**VALIDATION**

Validation works by setting the `validateCommand` option to a script which will be evaluated according to the `validate` option as follows:

- **none**: Default. This means no validation will occur.
- **focus**: `validateCommand` will be called when the spinbox receives or loses focus.
- **focusin**: `validateCommand` will be called when the spinbox receives focus.
- **focusout**: `validateCommand` will be called when the spinbox loses focus.
- **key**: `validateCommand` will be called when the spinbox is edited.
- **all**: `validateCommand` will be called for all above conditions.

It is possible to perform percent substitutions on the `validateCommand` and `invalidCommand`, just as you would in a `bind` script. The following substitutions are recognized:

- **%d**: Type of action: 1 for `insert`, 0 for `delete`, or −1 for focus, forced or `textvariable` validation.
- **%i**: Index of char string to be inserted/deleted, if any, otherwise −1.
- **%P**: The value of the spinbox should edition occur. If you are configuring the spinbox widget to have a new `textvariable`, this will be the value of that `textvariable`.
- **%s**: The current value of spinbox before edition.
- **%S**: The text string being inserted/deleted, if any. Otherwise it is an empty string.
- **%v**: The type of validation currently set.
- **%V**: The type of validation that triggered the callback (key, focusin, focusout, forced).
- **%W**: The name of the spinbox widget.
In general, the textVariable and validateCommand can be dangerous to mix. Any problems have been overcome so that using the validateCommand will not interfere with the traditional behavior of the spinbox widget. Using the textVariable for read-only purposes will never cause problems. The danger comes when you try set the textVariable to something that the validateCommand would not accept, which causes validate to become none (the invalidCommand will not be triggered). The same happens when an error occurs evaluating the validateCommand.

Primarily, an error will occur when the validateCommand or invalidCommand encounters an error in its script while evaluating or validateCommand does not return a valid Tcl boolean value. The validate option will also set itself to none when you edit the spinbox widget from within either the validateCommand or the invalidCommand. Such editions will override the one that was being validated. If you wish to edit the value of the widget during validation and still have the validate option set, you should include the command

```tcl
%W config -validate %v
```

in the validateCommand or invalidCommand (whichever one you were editing the spinbox widget from). It is also recommended to not set an associated textVariable during validation, as that can cause the spinbox widget to become out of sync with the textVariable.

**WIDGET COMMAND**

The spinbox command creates a new Tcl command whose name is pathName. This command may be used to invoke various operations on the widget. It has the following general form:

```tcl
pathName option ?arg arg ...?
```

Option and the args determine the exact behavior of the command.

Many of the widget commands for spinboxes take one or more indices as arguments. An index specifies a particular character in the spinbox's string, in any of the following ways:

- **number** Specifies the character as a numerical index, where 0 corresponds to the first character in the string.
- **anchor** Indicates the anchor point for the selection, which is set with the select from and select adjust widget commands.
- **end** Indicates the character just after the last one in the spinbox's string. This is equivalent to specifying a numerical index equal to the length of the spinbox's string.
- **insert** Indicates the character adjacent to and immediately following the insertion cursor.
- **sel.first** Indicates the first character in the selection. It is an error to use this form if the selection isn't in the spinbox window.
- **sel.last**
Indicates the character just after the last one in the selection. It is an error to use this form if the selection isn't in the spinbox window.

@number

In this form, number is treated as an x−coordinate in the spinbox's window; the character spanning that x−coordinate is used. For example, "@0" indicates the left−most character in the window.

Abbreviations may be used for any of the forms above, e.g. "e" or "sel.f". In general, out−of−range indices are automatically rounded to the nearest legal value.

The following commands are possible for spinbox widgets:

pathName bbox index

Returns a list of four numbers describing the bounding box of the character given by index. The first two elements of the list give the x and y coordinates of the upper−left corner of the screen area covered by the character (in pixels relative to the widget) and the last two elements give the width and height of the character, in pixels. The bounding box may refer to a region outside the visible area of the window.

pathName cget option

Returns the current value of the configuration option given by option. Option may have any of the values accepted by the spinbox command.

pathName configure ?option? ?value option value ...? 

Query or modify the configuration options of the widget. If no option is specified, returns a list describing all of the available options for pathName (see Tk_ConfigureInfo for information on the format of this list). If option is specified with no value, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no option is specified). If one or more option−value pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. Option may have any of the values accepted by the spinbox command.

pathName delete first ?last?

Delete one or more elements of the spinbox. First is the index of the first character to delete, and last is the index of the character just after the last one to delete. If last isn't specified it defaults to first+1, i.e. a single character is deleted. This command returns an empty string.

pathName get

Returns the spinbox's string.

pathName icursor index

Arrange for the insertion cursor to be displayed just before the character given by index. Returns an empty string.

pathName identify x y

Returns the name of the window element corresponding to coordinates x and y in the spinbox. Return value is one of: none, buttondown, buttonup, entry.

pathName index index

Returns the numerical index corresponding to index.

pathName insert index string

Insert the characters of string just before the character indicated by index. Returns an empty string.

pathName invoke element
Causes the specified element, either buttondown or buttonup, to be invoked, triggering the action associated with it.

`pathName scan option args`

This command is used to implement scanning on spinboxes. It has two forms, depending on `option`:

`pathName scan mark x`

Records `x` and the current view in the spinbox window; used in conjunction with later `scan dragto` commands. Typically this command is associated with a mouse button press in the widget. It returns an empty string.

`pathName scan dragto x`

This command computes the difference between its `x` argument and the `x` argument to the last `scan mark` command for the widget. It then adjusts the view left or right by 10 times the difference in x–coordinates. This command is typically associated with mouse motion events in the widget, to produce the effect of dragging the spinbox at high speed through the window. The return value is an empty string.

`pathName selection option arg`

This command is used to adjust the selection within a spinbox. It has several forms, depending on `option`:

`pathName selection adjust index`

Locate the end of the selection nearest to the character given by `index`, and adjust that end of the selection to be at `index` (i.e. including but not going beyond `index`). The other end of the selection is made the anchor point for future `select to` commands. If the selection isn't currently in the spinbox, then a new selection is created to include the characters between `index` and the most recent selection anchor point, inclusive. Returns an empty string.

`pathName selection clear`

Clear the selection if it is currently in this widget. If the selection isn't in this widget then the command has no effect. Returns an empty string.

`pathName selection element ?element?`

Sets or gets the currently selected element. If a spinbutton element is specified, it will be displayed depressed.

`pathName selection from index`

Set the selection anchor point to just before the character given by `index`. Doesn't change the selection. Returns an empty string.

`pathName selection present`

Returns 1 if there are characters selected in the spinbox, 0 if nothing is selected.

`pathName selection range start end`

Sets the selection to include the characters starting with the one indexed by `start` and ending with the one just before `end`. If `end` refers to the same character as `start` or an earlier one, then the spinbox's selection is cleared.

`pathName selection to index`

If `index` is before the anchor point, set the selection to the characters from `index` up to but not including the anchor point. If `index` is the same as the anchor point, do nothing. If `index` is after the anchor point, set the selection to the characters from the anchor point up to but not including `index`. The anchor point is determined by the most recent `select from` or `select`
adjust command in this widget. If the selection isn't in this widget then a new selection is created using the most recent anchor point specified for the widget. Returns an empty string.

pathName set ?string?

If string is specified, the spinbox will try and set it to this value, otherwise it just returns the spinbox's string. If validation is on, it will occur when setting the string.

pathName validate

This command is used to force an evaluation of the validateCommand independent of the conditions specified by the validate option. This is done by temporarily setting the validate option to all. It returns 0 or 1.

pathName xview args

This command is used to query and change the horizontal position of the text in the widget's window. It can take any of the following forms:

pathName xview

Returns a list containing two elements. Each element is a real fraction between 0 and 1; together they describe the horizontal span that is visible in the window. For example, if the first element is .2 and the second element is .6, 20% of the spinbox's text is off-screen to the left, the middle 40% is visible in the window, and 40% of the text is off-screen to the right. These are the same values passed to scrollbars via the –xscrollcommand option.

pathName xview index

Adjusts the view in the window so that the character given by index is displayed at the left edge of the window.

pathName xview moveto fraction

Adjusts the view in the window so that the character fraction of the way through the text appears at the left edge of the window. Fraction must be a fraction between 0 and 1.

pathName xview scroll number what

This command shifts the view in the window left or right according to number and what. Number must be an integer. What must be either units or pages or an abbreviation of one of these. If what is units, the view adjusts left or right by number average−width characters on the display; if it is pages then the view adjusts by number screenfuls. If number is negative then characters farther to the left become visible; if it is positive then characters farther to the right become visible.

DEFAULT BINDINGS

Tk automatically creates class bindings for spinboxes that give them the following default behavior. In the descriptions below, ``word” refers to a contiguous group of letters, digits, or ``_” characters, or any single character other than these.

[1] Clicking mouse button 1 positions the insertion cursor just before the character underneath the mouse cursor, sets the input focus to this widget, and clears any selection in the widget. Dragging with mouse button 1 strokes out a selection between the insertion cursor and the character under the mouse.

[2]
Double-clicking with mouse button 1 selects the word under the mouse and positions the insertion cursor at the beginning of the word. Dragging after a double click will stroke out a selection consisting of whole words.

Triple-clicking with mouse button 1 selects all of the text in the spinbox and positions the insertion cursor before the first character.

The ends of the selection can be adjusted by dragging with mouse button 1 while the Shift key is down; this will adjust the end of the selection that was nearest to the mouse cursor when button 1 was pressed. If the button is double-clicked before dragging then the selection will be adjusted in units of whole words.

Clicking mouse button 1 with the Control key down will position the insertion cursor in the spinbox without affecting the selection.

If any normal printing characters are typed in a spinbox, they are inserted at the point of the insertion cursor.

The view in the spinbox can be adjusted by dragging with mouse button 2. If mouse button 2 is clicked without moving the mouse, the selection is copied into the spinbox at the position of the mouse cursor.

If the mouse is dragged out of the spinbox on the left or right sides while button 1 is pressed, the spinbox will automatically scroll to make more text visible (if there is more text off-screen on the side where the mouse left the window).

The Left and Right keys move the insertion cursor one character to the left or right; they also clear any selection in the spinbox and set the selection anchor. If Left or Right is typed with the Shift key down, then the insertion cursor moves and the selection is extended to include the new character. Control–Left and Control–Right move the insertion cursor by words, and Control–Shift–Left and Control–Shift–Right move the insertion cursor by words and also extend the selection. Control–b and Control–f behave the same as Left and Right, respectively. Meta–b and Meta–f behave the same as Control–Left and Control–Right, respectively.

The Home key, or Control–a, will move the insertion cursor to the beginning of the spinbox and clear any selection in the spinbox. Shift–Home moves the insertion cursor to the beginning of the spinbox and also extends the selection to that point.

The End key, or Control–e, will move the insertion cursor to the end of the spinbox and clear any selection in the spinbox. Shift–End moves the cursor to the end and extends the selection to that point.

The Select key and Control–Space set the selection anchor to the position of the insertion cursor. They don't affect the current selection. Shift–Select and Control–Shift–Space adjust the selection to the current position of the insertion cursor, selecting from the anchor to the insertion cursor if there was not any selection previously.
Control−/ selects all the text in the spinbox.

Control−\ clears any selection in the spinbox.

The F16 key (labelled Copy on many Sun workstations) or Meta−w copies the selection in the widget to the clipboard, if there is a selection.

The F20 key (labelled Cut on many Sun workstations) or Control−w copies the selection in the widget to the clipboard and deletes the selection. If there is no selection in the widget then these keys have no effect.

The F18 key (labelled Paste on many Sun workstations) or Control−y inserts the contents of the clipboard at the position of the insertion cursor.

The Delete key deletes the selection, if there is one in the spinbox. If there is no selection, it deletes the character to the right of the insertion cursor.

The BackSpace key and Control−h delete the selection, if there is one in the spinbox. If there is no selection, it deletes the character to the left of the insertion cursor.

Control−d deletes the character to the right of the insertion cursor.

Meta−d deletes the word to the right of the insertion cursor.

Control−k deletes all the characters to the right of the insertion cursor.

Control−t reverses the order of the two characters to the right of the insertion cursor.

If the spinbox is disabled using the −state option, then the spinbox's view can still be adjusted and text in the spinbox can still be selected, but no insertion cursor will be displayed and no text modifications will take place.

The behavior of spinboxes can be changed by defining new bindings for individual widgets or by redefining the class bindings.

KEYWORDS

spinbox, entry, widget

Copyright 2000 Jeffrey Hobbs.
Copyright 2000 Ajuba Solutions.
Copyright 1995-1997 Roger E. Critchlow Jr.
NAME
text, tk_textCopy, tk_textCut, tk_textPaste – Create and manipulate text widgets

SYNOPSIS
text pathName ?options?
tk_textCopy pathName
tk_textCut pathName
tk_textPaste pathName

STANDARD OPTIONS
−background or −bg, background, Background
−borderwidth or −bd, borderWidth, BorderWidth
−cursor, cursor, Cursor
−exportselection, exportSelection, ExportSelection
−font, font, Font
−foreground or −fg, foreground, Foreground
−highlightbackground, highlightBackground, HighlightBackground
−highlightcolor, highlightColor, HighlightColor
−highlightthickness, highlightThickness, HighlightThickness
−insertbackground, insertBackground, Foreground
−insertborderwidth, insertBorderWidth, BorderWidth
−insertofftime, insertOffTime, OffTime
−inserton时间为, insertOnTime, OnTime
−insertwidth, insertWidth, InsertWidth
−padX, padX, Pad
−padY, padY, Pad
−relief, relief, Relief
−selectbackground, selectBackground, Foreground
−selectborderwidth, selectBorderWidth, BorderWidth
−selectforeground, selectForeground, Background
−setgrid, setGrid, SetGrid
−takefocus, takeFocus, TakeFocus
−xscrollcommand, xScrollCommand, ScrollCommand
−yscrollcommand, yScrollCommand, ScrollCommand

WIDGET–SPECIFIC OPTIONS
−autoseparators, autoSeparators, AutoSeparators
−height, height, Height
−maxundo, maxUndo, MaxUndo
−spacing1, spacing1, Spacing1
−spacing2, spacing2, Spacing2
−spacing3, spacing3, Spacing3
−state, state, State
−tabs, tabs, Tabs
undo, undo, Undo
−width, width, Width
−wrap, wrap, Wrap

DESCRIPTION

INDICES

line, char
@ x, y
end
mark
tag.first
tag.last
pathName
imageName
+ count chars
− count chars
+ count lines
− count lines
linestart
lineend
wordstart
wordend

TAGS

−background color
−bgstipple bitmap
−borderwidth pixels
−elide boolean
−fgstipple bitmap
−font fontName
−foreground color
−justify justify
−lmargin1 pixels
−lmargin2 pixels
−offset pixels
−overstrike boolean
−relief relief
−rmargin pixels
−spacing1 pixels
−spacing2 pixels
−spacing3 pixels
−tabs tabList
−underline boolean
−wrap mode

MARKS

EMBEDDED WINDOWS
−align where
-create script
-padx pixels
-pady pixels
-stretch boolean
-window pathName

EMBEDDED IMAGES
-align where
-image image
-name imageName
-padx pixels
-pady pixels

THE SELECTION
THE INSERTION CURSOR
THE MODIFIED FLAG
THE UNDO MECHANISM

WIDGET COMMAND

    pathName bbox index
    pathName cget option
    pathName compare index1 op index2
    pathName configure ?option? ?value option value ...?
    pathName debug ?boolean?
    pathName delete index1 ?index2 ...?
    pathName dlineinfo index
    pathName dump ?switches? index1 ?index2?
      -all
      -command command
      -image
      -mark
      -tag
      -text
      -window

    pathName edit option ?arg arg ...?
      pathName edit modified ?boolean?
      pathName edit redo
      pathName edit reset
      pathName edit separator
      pathName edit undo

    pathName get index1 ?index2 ...?
    pathName image option ?arg arg ...?
      pathName image cget index option
      pathName image configure index ?option value ...?
      pathName image create index ?option value ...?
      pathName image names

    pathName index index
    pathName insert index chars ?tagList chars tagList ...?
**pathName** `mark` `option` ?`arg` ...,?

`mark gravity` `markName` ?`direction`?

`mark names`

`mark next index`

`mark previous index`

`mark set` `markName` index

`mark unset` `markName` ...

**pathName** `scan` `option` `args`

`scan mark x y`

`scan dragto x y`

**pathName** `search` ?`switches`? `pattern` `index` `stopIndex`?

- `forwards`
- `backwards`
- `exact`
- `regexp`
- `nocase`
- `count` varName
- `elide`

**pathName** `see` `index`

**pathName** `tag` `option` ?`arg` ...,?

`tag add` `tagName` index1 ?index2 index1 index2 ...

`tag bind` `tagName` `sequence` ?`script`?

`tag cget` `tagName` `option`?

`tag configure` `tagName` ?`option` ?`value` ?`option value` ...

`tag delete` `tagName` `tagName` ...

`tag lower` `tagName` `belowThis`?

`tag names` ?`index`?

`tag nextrange` `tagName` index1 ?index2?

`tag prevrange` `tagName` index1 ?index2?

`tag raise` `tagName` ?`aboveThis`?

`tag ranges` `tagName`

`tag remove` `tagName` index1 ?index2 index1 index2 ...

**pathName** `window` `option` ?`arg` ...,?

`window cget` index `option`?

`window configure` index `option value` ...

`window create` index `option value` ...

`window names`

**pathName** `xview` `option` `args`

`xview`?

`xview moveto` `fraction`

`xview scroll` `number` `what`

**pathName** `yview` ?`args`?

`yview`?

`yview moveto` `fraction`
NAME
text, tk_textCopy, tk_textCut, tk_textPaste – Create and manipulate text widgets

SYNOPSIS

text pathName ?options?
tk_textCopy pathName
tk_textCut pathName
tk_textPaste pathName

STANDARD OPTIONS

−background or −bg, background, Background
−borderwidth or −bd, borderWidth, BorderWidth
−cursor, cursor, Cursor
−exportselection, exportSelection, ExportSelection
−font, font, Font
−foreground or −fg, foreground, Foreground
−highlightbackground, highlightBackground, HighlightBackground
−highlightcolor, highlightColor, HighlightColor
−highlightthickness, highlightThickness, HighlightThickness
−insertbackground, insertBackground, Foreground
−insertborderwidth, insertBorderWidth, BorderWidth
−insertofftime, insertOffTime, OffTime
−insertontime, insertOnTime, OnTime
−insertwidth, insertWidth, InsertWidth
−padx, padX, Pad
−pady, padY, Pad
−relief, relief, Relief
−selectbackground, selectBackground, Foreground
−selectborderwidth, selectBorderWidth, BorderWidth
−selectforeground, selectForeground, Background
−setgrid, setGrid, SetGrid
−takefocus, takeFocus, TakeFocus
−xscrollcommand, xScrollCommand, ScrollCommand
−yscrollcommand, yScrollCommand, ScrollCommand
WIDGET–SPECIFIC OPTIONS

Command–Line Name: −autoseparators
Database Name: autoSeparators
Database Class: AutoSeparators
Specifications a boolean that says whether separators are automatically inserted in the undo stack. Only meaningful when the −undo option is true.

Command–Line Name: −height
Database Name: height
Database Class: Height
Specifies the desired height for the window, in units of characters in the font given by the −font option. Must be at least one.

Command–Line Name: −maxundo
Database Name: maxUndo
Database Class: MaxUndo
Specifies the maximum number of compound undo actions on the undo stack. A zero or a negative value imply an unlimited undo stack.

Command–Line Name: −spacing1
Database Name: spacing1
Database Class: Spacing1
Requests additional space above each text line in the widget, using any of the standard forms for screen distances. If a line wraps, this option only applies to the first line on the display. This option may be overridden with −spacing1 options in tags.

Command–Line Name: −spacing2
Database Name: spacing2
Database Class: Spacing2
For lines that wrap (so that they cover more than one line on the display) this option specifies additional space to provide between the display lines that represent a single line of text. The value may have any of the standard forms for screen distances. This option may be overridden with −spacing2 options in tags.

Command–Line Name: −spacing3
Database Name: spacing3
Database Class: Spacing3
Requests additional space below each text line in the widget, using any of the standard forms for screen distances. If a line wraps, this option only applies to the last line on the display. This option may be overridden with −spacing3 options in tags.

Command–Line Name: −state
Database Name: state
Database Class: State
Specifies one of two states for the text: normal or disabled. If the text is disabled then characters may not be inserted or deleted and no insertion cursor will be displayed, even if the input focus is in the widget.

Command–Line Name: −tabs
Database Name: tabs
Database Class: Tabs
Specifies a set of tab stops for the window. The option's value consists of a list of screen distances giving the positions of the tab stops, each of which is a distance relative to the left edge of the widget (excluding borders, padding, etc). Each position may optionally be followed in the next list element by one of the keywords left, right, center, or numeric, which specifies how to justify text relative to the tab stop. Left is the default; it causes the text following the tab character to be positioned with its left edge at the tab position. Right means that the right edge of the text following the tab character is positioned at the tab position, and center means that the text is centered at the tab position. Numeric means that the decimal point in the text is positioned at the tab position; if there is no decimal point then the least significant digit of the number is positioned just to the left of the tab position; if there is no number in the text then the text is right-justified at the tab position. For example, –tabs {2c left 4c 6c center} creates three tab stops at two-centimeter intervals; the first two use left justification and the third uses center justification. If the list of tab stops does not have enough elements to cover all of the tabs in a text line, then Tk extrapolates new tab stops using the spacing and alignment from the last tab stop in the list. Tab distances must be strictly positive, and must always increase from one tab stop to the next (if not, an error is thrown). The value of the tabs option may be overridden by –tabs options in tags. If no –tabs option is specified, or if it is specified as an empty list, then Tk uses default tabs spaced every eight (average size) characters.

Command–Line Name: –undo
Database Name: undo
Database Class: Undo

Specifies a boolean that says whether the undo mechanism is active or not.

Command–Line Name: –width
Database Name: width
Database Class: Width

Specifies the desired width for the window in units of characters in the font given by the –font option. If the font doesn't have a uniform width then the width of the character “0” is used in translating from character units to screen units.

Command–Line Name: –wrap
Database Name: wrap
Database Class: Wrap

Specifies how to handle lines in the text that are too long to be displayed in a single line of the text's window. The value must be none or char or word. A wrap mode of none means that each line of text appears as exactly one line on the screen; extra characters that don't fit on the screen are not displayed. In the other modes each line of text will be broken up into several screen lines if necessary to keep all the characters visible. In char mode a screen line break may occur after any character; in word mode a line break will only be made at word boundaries.

DESCRIPTION

The text command creates a new window (given by the pathName argument) and makes it into a text widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the text such as its default background color and relief. The text command returns the path name of the new window.

A text widget displays one or more lines of text and allows that text to be edited. Text widgets support four
different kinds of annotations on the text, called tags, marks, embedded windows or embedded images. Tags allow different portions of the text to be displayed with different fonts and colors. In addition, Tcl commands can be associated with tags so that scripts are invoked when particular actions such as keystrokes and mouse button presses occur in particular ranges of the text. See TAGS below for more details.

The second form of annotation consists of floating markers in the text called "marks". Marks are used to keep track of various interesting positions in the text as it is edited. See MARKS below for more details.

The third form of annotation allows arbitrary windows to be embedded in a text widget. See EMBEDDED WINDOWS below for more details.

The fourth form of annotation allows Tk images to be embedded in a text widget. See EMBEDDED IMAGES below for more details.

The text widget also has a built-in undo/redo mechanism. See THE UNDO MECHANISM below for more details.

INDICES

Many of the widget commands for texts take one or more indices as arguments. An index is a string used to indicate a particular place within a text, such as a place to insert characters or one endpoint of a range of characters to delete. Indices have the syntax

\[
\text{base modifier modifier modifier ...}
\]

Where base gives a starting point and the modifiers adjust the index from the starting point (e.g. move forward or backward one character). Every index must contain a base, but the modifiers are optional.

The base for an index must have one of the following forms:

- \text{line.char} Indicates \text{char}'\text{th} character on line \text{line}. Lines are numbered from 1 for consistency with other UNIX programs that use this numbering scheme. Within a line, characters are numbered from 0. If \text{char} is end then it refers to the newline character that ends the line.

- \text{@x,y} Indicates the character that covers the pixel whose x and y coordinates within the text's window are \text{x} and \text{y}.

- \text{end} Indicates the end of the text (the character just after the last newline).

- \text{mark} Indicates the character just after the mark whose name is \text{mark}.

- \text{tag.first} Indicates the first character in the text that has been tagged with \text{tag}. This form generates an error if no characters are currently tagged with \text{tag}.

- \text{tag.last}
Indicates the character just after the last one in the text that has been tagged with tag. This form generates an error if no characters are currently tagged with tag.

pathName
Indicates the position of the embedded window whose name is pathName. This form generates an error if there is no embedded window by the given name.

imageName
Indicates the position of the embedded image whose name is imageName. This form generates an error if there is no embedded image by the given name.

If the base could match more than one of the above forms, such as a mark and imageName both having the same value, then the form earlier in the above list takes precedence. If modifiers follow the base index, each one of them must have one of the forms listed below. Keywords such as chars and wordend may be abbreviated as long as the abbreviation is unambiguous.

+ count chars
Adjust the index forward by count characters, moving to later lines in the text if necessary. If there are fewer than count characters in the text after the current index, then set the index to the last character in the text. Spaces on either side of count are optional.

− count chars
Adjust the index backward by count characters, moving to earlier lines in the text if necessary. If there are fewer than count characters in the text before the current index, then set the index to the first character in the text. Spaces on either side of count are optional.

+ count lines
Adjust the index forward by count lines, retaining the same character position within the line. If there are fewer than count lines after the line containing the current index, then set the index to refer to the same character position on the last line of the text. Then, if the line is not long enough to contain a character at the indicated character position, adjust the character position to refer to the last character of the line (the newline). Spaces on either side of count are optional.

− count lines
Adjust the index backward by count lines, retaining the same character position within the line. If there are fewer than count lines before the line containing the current index, then set the index to refer to the same character position on the first line of the text. Then, if the line is not long enough to contain a character at the indicated character position, adjust the character position to refer to the last character of the line (the newline). Spaces on either side of count are optional.

linestart
Adjust the index to refer to the first character on the line.

lineend
Adjust the index to refer to the last character on the line (the newline).

wordstart
Adjust the index to refer to the first character of the word containing the current index. A word consists of any number of adjacent characters that are letters, digits, or underscores, or a single character that is not one of these.

wordend
Adjust the index to refer to the character just after the last one of the word containing the current index. If the current index refers to the last character of the text then it is not modified.
If more than one modifier is present then they are applied in left–to–right order. For example, the index ``end – 1 chars'' refers to the next–to–last character in the text and ``insert wordstart – 1 c'' refers to the character just before the first one in the word containing the insertion cursor.

**TAGS**

The first form of annotation in text widgets is a tag. A tag is a textual string that is associated with some of the characters in a text. Tags may contain arbitrary characters, but it is probably best to avoid using the characters `` '' (space), +, or −: these characters have special meaning in indices, so tags containing them can't be used as indices. There may be any number of tags associated with characters in a text. Each tag may refer to a single character, a range of characters, or several ranges of characters. An individual character may have any number of tags associated with it.

A priority order is defined among tags, and this order is used in implementing some of the tag–related functions described below. When a tag is defined (by associating it with characters or setting its display options or binding commands to it), it is given a priority higher than any existing tag. The priority order of tags may be redefined using the ``pathName tag raise'' and ``pathName tag lower'' widget commands.

Tags serve three purposes in text widgets. First, they control the way information is displayed on the screen. By default, characters are displayed as determined by the background, font, and foreground options for the text widget. However, display options may be associated with individual tags using the ``pathName tag configure'' widget command. If a character has been tagged, then the display options associated with the tag override the default display style. The following options are currently supported for tags:

- **background color**
  
  *Color* specifies the background color to use for characters associated with the tag. It may have any of the forms accepted by *Tk_GetColor*.

- **bgstipple bitmap**
  
  *Bitmap* specifies a bitmap that is used as a stipple pattern for the background. It may have any of the forms accepted by *Tk_GetBitmap*. If *bitmap* hasn't been specified, or if it is specified as an empty string, then a solid fill will be used for the background.

- **borderwidth pixels**
  
  *Pixels* specifies the width of a 3–D border to draw around the background. It may have any of the forms accepted by *Tk_GetPixels*. This option is used in conjunction with the –relief option to give a 3–D appearance to the background for characters; it is ignored unless the –background option has been set for the tag.

- **elide boolean**
  
  *Elide* specifies whether the data should be elided. Elided data is not displayed and takes no space on screen, but further on behaves just as normal data.

- **fgstipple bitmap**
  
  *Bitmap* specifies a bitmap that is used as a stipple pattern when drawing text and other foreground information such as underlines. It may have any of the forms accepted by *Tk_GetBitmap*. If *bitmap* hasn't been specified, or if it is specified as an empty string, then a solid fill will be used.

- **font fontName**
  
  *FontName* is the name of a font to use for drawing characters. It may have any of the forms accepted
by \texttt{Tk\_GetFont}.

\texttt{−foreground color}

\textit{Color} specifies the color to use when drawing text and other foreground information such as underlines. It may have any of the forms accepted by \texttt{Tk\_GetColor}.

\texttt{−justify justify}

If the first character of a display line has a tag for which this option has been specified, then \texttt{justify} determines how to justify the line. It must be one of \texttt{left}, \texttt{right}, or \texttt{center}. If a line wraps, then the justification for each line on the display is determined by the first character of that display line.

\texttt{−lmargin1 pixels}

If the first character of a text line has a tag for which this option has been specified, then \texttt{pixels} specifies how much the line should be indented from the left edge of the window. \texttt{Pixels} may have any of the standard forms for screen distances. If a line of text wraps, this option only applies to the first line on the display; the \texttt{−lmargin2} option controls the indentation for subsequent lines.

\texttt{−lmargin2 pixels}

If the first character of a display line has a tag for which this option has been specified, and if the display line is not the first for its text line (i.e., the text line has wrapped), then \texttt{pixels} specifies how much the line should be indented from the left edge of the window. \texttt{Pixels} may have any of the standard forms for screen distances. This option is only used when wrapping is enabled, and it only applies to the second and later display lines for a text line.

\texttt{−offset pixels}

\texttt{Pixels} specifies an amount by which the text's baseline should be offset vertically from the baseline of the overall line, in pixels. For example, a positive offset can be used for superscripts and a negative offset can be used for subscripts. \texttt{Pixels} may have any of the standard forms for screen distances.

\texttt{−overstrike boolean}

Specifies whether or not to draw a horizontal rule through the middle of characters. \texttt{Boolean} may have any of the forms accepted by \texttt{Tk\_GetBoolean}.

\texttt{−relief relief}

\textit{Relief} specifies the 3-D relief to use for drawing backgrounds, in any of the forms accepted by \texttt{Tk\_GetRelief}. This option is used in conjunction with the \texttt{−borderwidth} option to give a 3-D appearance to the background for characters; it is ignored unless the \texttt{−background} option has been set for the tag.

\texttt{−rmargin pixels}

If the first character of a display line has a tag for which this option has been specified, then \texttt{pixels} specifies how wide a margin to leave between the end of the line and the right edge of the window. \texttt{Pixels} may have any of the standard forms for screen distances. This option is only used when wrapping is enabled. If a text line wraps, the right margin for each line on the display is determined by the first character of that display line.

\texttt{−spacing1 pixels}

\texttt{Pixels} specifies how much additional space should be left above each text line, using any of the standard forms for screen distances. If a line wraps, this option only applies to the first line on the display.

\texttt{−spacing2 pixels}

For lines that wrap, this option specifies how much additional space to leave between the display lines for a single text line. \texttt{Pixels} may have any of the standard forms for screen distances.

\texttt{−spacing3 pixels}
Pixels specifies how much additional space should be left below each text line, using any of the standard forms for screen distances. If a line wraps, this option only applies to the last line on the display.

-tabs tabList

TabList specifies a set of tab stops in the same form as for the -tabs option for the text widget. This option only applies to a display line if it applies to the first character on that display line. If this option is specified as an empty string, it cancels the option, leaving it unspecified for the tag (the default). If the option is specified as a non-empty string that is an empty list, such as -tags { }, then it requests default 8-character tabs as described for the tags widget option.

-underline boolean

Boolean specifies whether or not to draw an underline underneath characters. It may have any of the forms accepted by Tk_GetBoolean.

-wrap mode

Mode specifies how to handle lines that are wider than the text's window. It has the same legal values as the -wrap option for the text widget: none, char, or word. If this tag option is specified, it overrides the -wrap option for the text widget.

If a character has several tags associated with it, and if their display options conflict, then the options of the highest priority tag are used. If a particular display option hasn't been specified for a particular tag, or if it is specified as an empty string, then that option will never be used; the next-highest-priority tag's option will be used instead. If no tag specifies a particular display option, then the default style for the widget will be used.

The second purpose for tags is event bindings. You can associate bindings with a tag in much the same way you can associate bindings with a widget class: whenever particular X events occur on characters with the given tag, a given Tcl command will be executed. Tag bindings can be used to give behaviors to ranges of characters; among other things, this allows hypertext-like features to be implemented. For details, see the description of the tag bind widget command below.

The third use for tags is in managing the selection. See THE SELECTION below.

MARKS

The second form of annotation in text widgets is a mark. Marks are used for remembering particular places in a text. They are something like tags, in that they have names and they refer to places in the file, but a mark isn't associated with particular characters. Instead, a mark is associated with the gap between two characters. Only a single position may be associated with a mark at any given time. If the characters around a mark are deleted the mark will still remain; it will just have new neighbor characters. In contrast, if the characters containing a tag are deleted then the tag will no longer have an association with characters in the file. Marks may be manipulated with the `pathName mark' widget command, and their current locations may be determined by using the mark name as an index in widget commands.

Each mark also has a "gravity", which is either left or right. The gravity for a mark specifies what happens to the mark when text is inserted at the point of the mark. If a mark has left gravity, then the mark is treated as if it were attached to the character on its left, so the mark will remain to the left of any text inserted at the mark position. If the mark has right gravity, new text inserted at the mark position will appear to the left of the mark

MARKS
(so that the mark remains rightmost). The gravity for a mark defaults to right.

The name space for marks is different from that for tags: the same name may be used for both a mark and a tag, but they will refer to different things.

Two marks have special significance. First, the mark insert is associated with the insertion cursor, as described under THE INSERTION CURSOR below. Second, the mark current is associated with the character closest to the mouse and is adjusted automatically to track the mouse position and any changes to the text in the widget (one exception: current is not updated in response to mouse motions if a mouse button is down; the update will be deferred until all mouse buttons have been released). Neither of these special marks may be deleted.

**EMBEDDED WINDOWS**

The third form of annotation in text widgets is an embedded window. Each embedded window annotation causes a window to be displayed at a particular point in the text. There may be any number of embedded windows in a text widget, and any widget may be used as an embedded window (subject to the usual rules for geometry management, which require the text window to be the parent of the embedded window or a descendant of its parent). The embedded window's position on the screen will be updated as the text is modified or scrolled, and it will be mapped and unmapped as it moves into and out of the visible area of the text widget. Each embedded window occupies one character's worth of index space in the text widget, and it may be referred to either by the name of its embedded window or by its position in the widget's index space. If the range of text containing the embedded window is deleted then the window is destroyed.

When an embedded window is added to a text widget with the window create widget command, several configuration options may be associated with it. These options may be modified later with the window configure widget command. The following options are currently supported:

- **-align where**
  If the window is not as tall as the line in which it is displayed, this option determines where the window is displayed in the line. Where must have one of the values top (align the top of the window with the top of the line), center (center the window within the range of the line), bottom (align the bottom of the window with the bottom of the line's area), or baseline (align the bottom of the window with the baseline of the line).

- **-create script**
  Specifies a Tcl script that may be evaluated to create the window for the annotation. If no -window option has been specified for the annotation this script will be evaluated when the annotation is about to be displayed on the screen. Script must create a window for the annotation and return the name of that window as its result. If the annotation's window should ever be deleted, script will be evaluated again the next time the annotation is displayed.

- **-padx pixels**
  Pixels specifies the amount of extra space to leave on each side of the embedded window. It may have any of the usual forms defined for a screen distance.

- **-pady pixels**
  Pixels specifies the amount of extra space to leave on the top and on the bottom of the embedded
window. It may have any of the usual forms defined for a screen distance.

\texttt{--stretch boolean}

If the requested height of the embedded window is less than the height of the line in which it is displayed, this option can be used to specify whether the window should be stretched vertically to fill its line. If the \texttt{--pady} option has been specified as well, then the requested padding will be retained even if the window is stretched.

\texttt{--window pathName}

Specifies the name of a window to display in the annotation.

EMBEDDED IMAGES

The final form of annotation in text widgets is an embedded image. Each embedded image annotation causes an image to be displayed at a particular point in the text. There may be any number of embedded images in a text widget, and a particular image may be embedded in multiple places in the same text widget. The embedded image's position on the screen will be updated as the text is modified or scrolled. Each embedded image occupies one character's worth of index space in the text widget, and it may be referred to either by its position in the widget's index space, or the name it is assigned when the image is inserted into the text widget with \texttt{image create}. If the range of text containing the embedded image is deleted then that copy of the image is removed from the screen.

When an embedded image is added to a text widget with the \texttt{image create} widget command, a name unique to this instance of the image is returned. This name may then be used to refer to this image instance. The name is taken to be the value of the \texttt{--name} option (described below). If the \texttt{--name} option is not provided, the \texttt{--image} name is used instead. If the \texttt{imageName} is already in use in the text widget, then \texttt{#nn} is added to the end of the \texttt{imageName}, where \texttt{nn} is an arbitrary integer. This insures the \texttt{imageName} is unique. Once this name is assigned to this instance of the image, it does not change, even though the \texttt{--image} or \texttt{--name} values can be changed with \texttt{image configure}.

When an embedded image is added to a text widget with the \texttt{image create} widget command, several configuration options may be associated with it. These options may be modified later with the \texttt{image configure} widget command. The following options are currently supported:

\texttt{--align where}

If the image is not as tall as the line in which it is displayed, this option determines where the image is displayed in the line. \texttt{Where} must have one of the values \texttt{top} (align the top of the image with the top of the line), \texttt{center} (center the image within the range of the line), \texttt{bottom} (align the bottom of the image with the bottom of the line's area), or \texttt{baseline} (align the bottom of the image with the baseline of the line).

\texttt{--image image}

Specifies the name of the Tk image to display in the annotation. If \texttt{image} is not a valid Tk image, then an error is returned.

\texttt{--name ImageName}

Specifies the name by which this image instance may be referenced in the text widget. If \texttt{ImageName} is not supplied, then the name of the Tk image is used instead. If the \texttt{imageName} is already in use, \texttt{#nn} is appended to the end of the name as described above.
−padx pixels
    Pixels specifies the amount of extra space to leave on each side of the embedded image. It may have
    any of the usual forms defined for a screen distance.
−pady pixels
    Pixels specifies the amount of extra space to leave on the top and on the bottom of the embedded
    image. It may have any of the usual forms defined for a screen distance.

THE SELECTION

Selection support is implemented via tags. If the exportSelection option for the text widget is true then the sel
tag will be associated with the selection:

[1] Whenever characters are tagged with sel the text widget will claim ownership of the selection.
[2] Attempts to retrieve the selection will be serviced by the text widget, returning all the characters with
    the sel tag.
[3] If the selection is claimed away by another application or by another window within this application,
    then the sel tag will be removed from all characters in the text.
[4] Whenever the sel tag range changes a virtual event <<Selection>> is generated.

The sel tag is automatically defined when a text widget is created, and it may not be deleted with the
`pathName tag delete` widget command. Furthermore, the selectBackground, selectBorderWidth, and
selectForeground options for the text widget are tied to the −background, −borderwidth, and −foreground
options for the sel tag; changes in either will automatically be reflected in the other.

THE INSERTION CURSOR

The mark named insert has special significance in text widgets. It is defined automatically when a text widget
is created and it may not be unset with the `pathName mark unset` widget command. The insert mark
represents the position of the insertion cursor, and the insertion cursor will automatically be drawn at this
point whenever the text widget has the input focus.

THE MODIFIED FLAG

The text widget can keep track of changes to the content of the widget by means of the modified flag.
Inserting or deleting text will set this flag. The flag can be queried, set and cleared programmatically as well.
Whenever the flag changes state a <<Modified>> virtual event is generated. See the edit modified widget
command for more details.
THE UNDO MECHANISM

The text widget has an unlimited undo and redo mechanism (when the `-undo` widget option is true) which records every insert and delete action on a stack.

Boundaries (called "separators") are inserted between edit actions. The purpose of these separators is to group inserts, deletes and replaces into one compound edit action. When undoing a change everything between two separators will be undone. The undone changes are then moved to the redo stack, so that an undone edit can be redone again. The redo stack is cleared whenever new edit actions are recorded on the undo stack. The undo and redo stacks can be cleared to keep their depth under control.

Separators are inserted automatically when the `-autoseparators` widget option is true. You can insert separators programmatically as well. If a separator is already present at the top of the undo stack no other will be inserted. That means that two separators on the undo stack are always separated by at least one insert or delete action.

The undo mechanism is also linked to the modified flag. This means that undoing or redoing changes can take a modified text widget back to the unmodified state or vice versa. The modified flag will be set automatically to the appropriate state. This automatic coupling does not work when the modified flag has been set by the user, until the flag has been reset again.

See below for the `edit` widget command that controls the undo mechanism.

WIDGET COMMAND

The `text` command creates a new Tcl command whose name is the same as the path name of the text's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

`PathName` is the name of the command, which is the same as the text widget's path name. `Option` and the `args` determine the exact behavior of the command. The following commands are possible for text widgets:

- `pathName bbox index`  
  Returns a list of four elements describing the screen area of the character given by `index`. The first two elements of the list give the x and y coordinates of the upper–left corner of the area occupied by the character, and the last two elements give the width and height of the area. If the character is only partially visible on the screen, then the return value reflects just the visible part. If the character is not visible on the screen then the return value is an empty list.

- `pathName cget option`  
  Returns the current value of the configuration option given by `option`. `Option` may have any of the values accepted by the `text` command.

- `pathName compare index1 op index2`  
  Compares the indices given by `index1` and `index2` according to the relational operator given by `op`,
and returns 1 if the relationship is satisfied and 0 if it isn't. \(Op\) must be one of the operators \(<\), \(<=\), \(==\), \(>=\), \(>\), or \(!=\). If \(op\) is \(==\) then 1 is returned if the two indices refer to the same character, if \(op\) is \(<\) then 1 is returned if \(index1\) refers to an earlier character in the text than \(index2\), and so on.

**pathName configure ?option? ?value option value ...?**

Query or modify the configuration options of the widget. If no \(option\) is specified, returns a list describing all of the available options for \(pathName\) (see \texttt{Tk\_ConfigureInfo} for information on the format of this list). If \(option\) is specified with no \(value\), then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no \(option\) is specified). If one or more \(option\)−\(value\) pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. \(Option\) may have any of the values accepted by the \texttt{text} command.

**pathName debug ?boolean?**

If \(boolean\) is specified, then it must have one of the true or false values accepted by \texttt{Tcl\_GetBoolean}. If the value is a true one then internal consistency checks will be turned on in the B−tree code associated with text widgets. If \(boolean\) has a false value then the debugging checks will be turned off. In either case the command returns an empty string. If \(boolean\) is not specified then the command returns \texttt{on} or \texttt{off} to indicate whether or not debugging is turned on. There is a single debugging switch shared by all text widgets: turning debugging on or off in any widget turns it on or off for all widgets. For widgets with large amounts of text, the consistency checks may cause a noticeable slow−down.

When debugging is turned on, the drawing routines of the text widget set the global variables \texttt{tk\_textRedraw} and \texttt{tk\_textRelayout} to the lists of indices that are redrawn. The values of these variables are tested by Tk's test suite.

**pathName delete index1 ?index2 ...?**

Delete a range of characters from the text. If both \(index1\) and \(index2\) are specified, then delete all the characters starting with the one given by \(index1\) and stopping just before \(index2\) (i.e. the character at \(index2\) is not deleted). If \(index2\) doesn't specify a position later in the text than \(index1\) then no characters are deleted. If \(index2\) isn't specified then the single character at \(index1\) is deleted. It is not allowable to delete characters in a way that would leave the text without a newline as the last character. The command returns an empty string. If more indices are given, multiple ranges of text will be deleted. All indices are first checked for validity before any deletions are made. They are sorted and the text is removed from the last range to the first range to deleted text does not cause an undesired index shifting side−effects. If multiple ranges with the same start index are given, then the longest range is used. If overlapping ranges are given, then they will be merged into spans that do not cause deletion of text outside the given ranges due to text shifted during deletion.

**pathName dlineinfo index**

Returns a list with five elements describing the area occupied by the display line containing \(index\). The first two elements of the list give the \(x\) and \(y\) coordinates of the upper−left corner of the area occupied by the line, the third and fourth elements give the width and height of the area, and the fifth element gives the position of the baseline for the line, measured down from the top of the area. All of this information is measured in pixels. If the current wrap mode is \texttt{none} and the line extends beyond the boundaries of the window, the area returned reflects the entire area of the line, including the portions that are out of the window. If the line is shorter than the full width of the window then the area returned reflects just the portion of the line that is occupied by characters and embedded
windows. If the display line containing index is not visible on the screen then the return value is an empty list.

```
pathName dump ?switches? index1 ?index2?
```

Return the contents of the text widget from index1 up to, but not including index2, including the text and information about marks, tags, and embedded windows. If index2 is not specified, then it defaults to one character past index1. The information is returned in the following format:

```
key1 value1 index1 key2 value2 index2 ...
```

The possible key values are text, mark, tagon, tagoff, and window. The corresponding value is the text, mark name, tag name, or window name. The index information is the index of the start of the text, the mark, the tag transition, or the window. One or more of the following switches (or abbreviations thereof) may be specified to control the dump:

```
−all
```

Return information about all elements: text, marks, tags, images and windows. This is the default.

```
−command command
```

Instead of returning the information as the result of the dump operation, invoke the command on each element of the text widget within the range. The command has three arguments appended to it before it is evaluated: the key, value, and index.

```
−image
```

Include information about images in the dump results.

```
−mark
```

Include information about marks in the dump results.

```
−tag
```

Include information about tag transitions in the dump results. Tag information is returned as tagon and tagoff elements that indicate the begin and end of each range of each tag, respectively.

```
−text
```

Include information about text in the dump results. The value is the text up to the next element or the end of range indicated by index2. A text element does not span newlines. A multi–line block of text that contains no marks or tag transitions will still be dumped as a set of text segments that each end with a newline. The newline is part of the value.

```
−window
```

Include information about embedded windows in the dump results. The value of a window is its Tk pathname, unless the window has not been created yet. (It must have a create script.) In this case an empty string is returned, and you must query the window by its index position to get more information.

```
pathName edit option ?arg arg ...?
```

This command controls the undo mechanism and the modified flag. The exact behavior of the command depends on the option argument that follows the edit argument. The following forms of the command are currently supported:

THE UNDO MECHANISM 645
pathName edit modified ?boolean?

If boolean is not specified, returns the modified flag of the widget. The insert, delete, edit undo and edit redo commands or the user can set or clear the modified flag. If boolean is specified, sets the modified flag of the widget to boolean.

pathName edit redo

When the −undo option is true, reappplies the last undone edits provided no other edits were done since then. Generates an error when the redo stack is empty. Does nothing when the −undo option is false.

pathName edit reset

Clears the undo and redo stacks.

pathName edit separator

Inserts a separator (boundary) on the undo stack. Does nothing when the −undo option is false.

pathName edit undo

Undoes the last edit action when the −undo option is true. An edit action is defined as all the insert and delete commands that are recorded on the undo stack in between two separators. Generates an error when the undo stack is empty. Does nothing when the −undo option is false.

pathName get index1 ?index2 ...?

Return a range of characters from the text. The return value will be all the characters in the text starting with the one whose index is index1 and ending just before the one whose index is index2 (the character at index2 will not be returned). If index2 is omitted then the single character at index1 is returned. If there are no characters in the specified range (e.g. index1 is past the end of the file or index2 is less than or equal to index1) then an empty string is returned. If the specified range contains embedded windows, no information about them is included in the returned string. If multiple index pairs are given, multiple ranges of text will be returned in a list. Invalid ranges will not be represented with empty strings in the list. The ranges are returned in the order passed to get.

pathName image option ?arg arg ...?

This command is used to manipulate embedded images. The behavior of the command depends on the option argument that follows the tag argument. The following forms of the command are currently supported:

pathName image cget index option

Returns the value of a configuration option for an embedded image. Index identifies the embedded image, and option specifies a particular configuration option, which must be one of the ones listed in the section EMBEDDED IMAGES.

pathName image configure index ?option value ...?

Query or modify the configuration options for an embedded image. If no option is specified, returns a list describing all of the available options for the embedded image at index (see Tk_ConfigureInfo for information on the format of this list). If option is specified with no value, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no option is specified). If one or more option−value pairs are specified, then the command modifies the given option(s) to have the given value(s); in this case the command returns an empty string. See EMBEDDED IMAGES for information on the options that are supported.
The `image create` command creates a new image annotation, which will appear in the text at the position given by `index`. Any number of `option`-`value` pairs may be specified to configure the annotation. Returns a unique identifier that may be used as an index to refer to this image. See **EMBEDDED IMAGES** for information on the options that are supported, and a description of the identifier returned.

The `image names` command returns a list whose elements are the names of all image instances currently embedded in `window`.

The `index` command returns the position corresponding to `index` in the form `line.char` where `line` is the line number and `char` is the character number. `Index` may have any of the forms described under **INDICES** above.

The `insert` command inserts all of the `chars` arguments just before the character at `index`. If `index` refers to the end of the text (the character after the last newline) then the new text is inserted just before the last newline instead. If there is a single `chars` argument and no `tagList`, then the new text will receive any tags that are present on both the character before and the character after the insertion point; if a tag is present on only one of these characters then it will not be applied to the new text. If `tagList` is specified then it consists of a list of tag names; the new characters will receive all of the tags in this list and no others, regardless of the tags present around the insertion point. If multiple `chars–tagList` argument pairs are present, they produce the same effect as if a separate `insert` widget command had been issued for each pair, in order. The last `tagList` argument may be omitted.

The `mark` command is used to manipulate marks. The exact behavior of the command depends on the `option` argument that follows the `mark` argument. The following forms of the command are currently supported:

- **`mark gravity`**
  - If `direction` is not specified, returns `left` or `right` to indicate which of its adjacent characters `markName` is attached to. If `direction` is specified, it must be `left` or `right`; the gravity of `markName` is set to the given value.

- **`mark names`**
  - Returns a list whose elements are the names of all the marks that are currently set.

- **`mark next`**
  - Returns the name of the next mark at or after `index`. If `index` is specified in numerical form, then the search for the next mark begins at that index. If `index` is the name of a mark, then the search for the next mark begins immediately after that mark. This can still return a mark at the same position if there are multiple marks at the same index. These semantics mean that the `mark next` operation can be used to step through all the marks in a text widget in the same order as the mark information returned by the `dump` operation. If a mark has been set to the special `end` index, then it appears to be `after end` with respect to the `mark next` operation. An empty string is returned if there are no marks after `index`.

- **`mark previous`**
  - Returns the name of the mark at or before `index`. If `index` is specified in numerical form, then the search for the previous mark begins with the character just before that index. If `index` is
the name of a mark, then the search for the next mark begins immediately before that mark. This can still return a mark at the same position if there are multiple marks at the same index. These semantics mean that the mark previous operation can be used to step through all the marks in a text widget in the reverse order as the mark information returned by the dump operation. An empty string is returned if there are no marks before index.

`pathName mark set markName index`

Sets the mark named markName to a position just before the character at index. If markName already exists, it is moved from its old position; if it doesn't exist, a new mark is created. This command returns an empty string.

`pathName mark unset markName ?markName markName ...?

Remove the mark corresponding to each of the markName arguments. The removed marks will not be usable in indices and will not be returned by future calls to `pathName mark names`. This command returns an empty string.

`pathName scan option args`

This command is used to implement scanning on texts. It has two forms, depending on option:

`pathName scan mark x y`

Records x and y and the current view in the text window, for use in conjunction with later scan dragto commands. Typically this command is associated with a mouse button press in the widget. It returns an empty string.

`pathName scan dragto x y`

This command computes the difference between its x and y arguments and the x and y arguments to the last scan mark command for the widget. It then adjusts the view by 10 times the difference in coordinates. This command is typically associated with mouse motion events in the widget, to produce the effect of dragging the text at high speed through the window. The return value is an empty string.

`pathName search ?switches? pattern index ?stopIndex?`

Searches the text in pathName starting at index for a range of characters that matches pattern. If a match is found, the index of the first character in the match is returned as result; otherwise an empty string is returned. One or more of the following switches (or abbreviations thereof) may be specified to control the search:

−forwards

The search will proceed forward through the text, finding the first matching range starting at or after the position given by index. This is the default.

−backwards

The search will proceed backward through the text, finding the matching range closest to index whose first character is before index.

−exact

Use exact matching: the characters in the matching range must be identical to those in pattern. This is the default.

−regexp

Treat pattern as a regular expression and match it against the text using the rules for regular expressions (see the regexp command for details).

−nocase
Ignore case differences between the pattern and the text.

**–count varName**

The argument following **–count** gives the name of a variable; if a match is found, the number of index positions between beginning and end of the matching range will be stored in the variable. If there are no embedded images or windows in the matching range (and there are no elided characters if **–elide** is not given), this is equivalent to the number of characters matched. In either case, the range `matchIdx to matchIdx + $count chars` will return the entire matched text.

**–elide**

Find elided (hidden) text as well. By default only displayed text is searched.

**–**

This switch has no effect except to terminate the list of switches: the next argument will be treated as `pattern` even if it starts with **–**.

The matching range must be entirely within a single line of text. For regular expression matching the newlines are removed from the ends of the lines before matching: use the `$` feature in regular expressions to match the end of a line. For exact matching the newlines are retained. If `stopIndex` is specified, the search stops at that index: for forward searches, no match at or after `stopIndex` will be considered; for backward searches, no match earlier in the text than `stopIndex` will be considered. If `stopIndex` is omitted, the entire text will be searched: when the beginning or end of the text is reached, the search continues at the other end until the starting location is reached again; if `stopIndex` is specified, no wrap-around will occur.

**`pathName see index`**

Adjusts the view in the window so that the character given by `index` is completely visible. If `index` is already visible then the command does nothing. If `index` is a short distance out of view, the command adjusts the view just enough to make `index` visible at the edge of the window. If `index` is far out of view, then the command centers `index` in the window.

**`pathName tag option ?arg arg ...?`**

This command is used to manipulate tags. The exact behavior of the command depends on the `option` argument that follows the `tag` argument. The following forms of the command are currently supported:

**`pathName tag add tagName index1 ?index2 index1 index2 ...?`**

Associate the tag `tagName` with all of the characters starting with `index1` and ending just before `index2` (the character at `index2` isn't tagged). A single command may contain any number of `index1–index2` pairs. If the last `index2` is omitted then the single character at `index1` is tagged. If there are no characters in the specified range (e.g. `index1` is past the end of the file or `index2` is less than or equal to `index1`) then the command has no effect.

**`pathName tag bind tagName ?sequence? ?script?`**

This command associates `script` with the tag given by `tagName`. Whenever the event sequence given by `sequence` occurs for a character that has been tagged with `tagName`, the script will be invoked. This widget command is similar to the `bind` command except that it operates on characters in a text rather than entire widgets. See the `bind` manual entry for complete details on the syntax of `sequence` and the substitutions performed on `script` before invoking it. If all arguments are specified then a new binding is created, replacing any existing binding for the same `sequence` and `tagName` (if the first character of `script` is ```+``` then `script` augments an
existing binding rather than replacing it). In this case the return value is an empty string. If
*script* is omitted then the command returns the *script* associated with *tagName* and *sequence*
(an error occurs if there is no such binding). If both *script* and *sequence* are omitted then the
command returns a list of all the sequences for which bindings have been defined for
*tagName*.

The only events for which bindings may be specified are those related to the mouse and
keyboard (such as *Enter*, *Leave*, *ButtonPress*, *Motion*, and *KeyPress*) or virtual events.
Event bindings for a text widget use the *current* mark described under *MARKS* above. An
*Enter* event triggers for a tag when the tag first becomes present on the current character, and
a *Leave* event triggers for a tag when it ceases to be present on the current character. *Enter*
and *Leave* events can happen either because the *current* mark moved or because the
character at that position changed. Note that these events are different than *Enter* and *Leave*
events for windows. Mouse and keyboard events are directed to the current character. If a
virtual event is used in a binding, that binding can trigger only if the virtual event is defined
by an underlying mouse−related or keyboard−related event.

It is possible for the current character to have multiple tags, and for each of them to have a
binding for a particular event sequence. When this occurs, one binding is invoked for each
tag, in order from lowest−priority to highest priority. If there are multiple matching bindings
for a single tag, then the most specific binding is chosen (see the manual entry for the *bind*
cmd command for details). *continue* and *break* commands within binding scripts are processed in
the same way as for bindings created with the *bind* command.

If bindings are created for the widget as a whole using the *bind* command, then those
bindings will supplement the tag bindings. The tag bindings will be invoked first, followed by
bindings for the window as a whole.

```
pathName tag cget tagName option
```

This command returns the current value of the option named *option* associated with the tag
given by *tagName*. *Option* may have any of the values accepted by the *tag configure* widget
command.

```
pathName tag configure tagName ?option? ?value? ?option value ...?
```

This command is similar to the *configure* widget command except that it modifies options
associated with the tag given by *tagName* instead of modifying options for the overall text
widget. If no *option* is specified, the command returns a list describing all of the available
options for *tagName* (see *Tk_ConfigureInfo* for information on the format of this list). If
*option* is specified with no *value*, then the command returns a list describing the one named
option (this list will be identical to the corresponding sublist of the value returned if no *option*
is specified). If one or more *option−value* pairs are specified, then the command modifies the
given option(s) to have the given value(s) in *tagName*; in this case the command returns an
empty string. See *TAGS* above for details on the options available for tags.

```
pathName tag delete tagName ?tagName ...?
```

Deletes all tag information for each of the *tagName* arguments. The command removes the
tags from all characters in the file and also deletes any other information associated with the
tags, such as bindings and display information. The command returns an empty string.
**pathName tag lower **tagName **?belowThis?**
Changes the priority of tag **tagName** so that it is just lower in priority than the tag whose name is **belowThis**. If **belowThis** is omitted, then **tagName**'s priority is changed to make it lowest priority of all tags.

**pathName tag names **?index?**
Returns a list whose elements are the names of all the tags that are active at the character position given by **index**. If **index** is omitted, then the return value will describe all of the tags that exist for the text (this includes all tags that have been named in a ``pathName tag'' widget command but haven't been deleted by a ``pathName tag delete'' widget command, even if no characters are currently marked with the tag). The list will be sorted in order from lowest priority to highest priority.

**pathName tag nextrange **tagName **index1 **?index2?**
This command searches the text for a range of characters tagged with **tagName** where the first character of the range is no earlier than the character at **index1** and no later than the character just before **index2** (a range starting at **index2** will not be considered). If several matching ranges exist, the first one is chosen. The command's return value is a list containing two elements, which are the index of the first character of the range and the index of the character just after the last one in the range. If no matching range is found then the return value is an empty string. If **index2** is not given then it defaults to the end of the text.

**pathName tag prevrange **tagName **index1 **?index2?**
This command searches the text for a range of characters tagged with **tagName** where the first character of the range is before the character at **index1** and no earlier than the character at **index2** (a range starting at **index2** will be considered). If several matching ranges exist, the one closest to **index1** is chosen. The command's return value is a list containing two elements, which are the index of the first character of the range and the index of the character just after the last one in the range. If no matching range is found then the return value is an empty string. If **index2** is not given then it defaults to the beginning of the text.

**pathName tag raise **tagName **?aboveThis?**
Changes the priority of tag **tagName** so that it is just higher in priority than the tag whose name is **aboveThis**. If **aboveThis** is omitted, then **tagName**'s priority is changed to make it highest priority of all tags.

**pathName tag ranges **tagName**
Returns a list describing all of the ranges of text that have been tagged with **tagName**. The first two elements of the list describe the first tagged range in the text, the next two elements describe the second range, and so on. The first element of each pair contains the index of the first character of the range, and the second element of the pair contains the index of the character just after the last one in the range. If there are no characters tagged with **tag** then an empty string is returned.

**pathName tag remove **tagName **index1 **?index2 index1 index2 ...?**
Remove the tag **tagName** from all of the characters starting at **index1** and ending just before **index2** (the character at **index2** isn't affected). A single command may contain any number of **index1**−**index2** pairs. If the last **index2** is omitted then the single character at **index1** is tagged. If there are no characters in the specified range (e.g. **index1** is past the end of the file or **index2** is less than or equal to **index1**) then the command has no effect. This command returns an empty string.
pathName window option ?arg arg ...?
This command is used to manipulate embedded windows. The behavior of the command depends on the option argument that follows the tag argument. The following forms of the command are currently supported:

pathName window cget index option
Returns the value of a configuration option for an embedded window. Index identifies the embedded window, and option specifies a particular configuration option, which must be one of the ones listed in the section EMBEDDED WINDOWS.

pathName window configure index ?option value ...?
Query or modify the configuration options for an embedded window. If no option is specified, returns a list describing all of the available options for the embedded window at index (see Tk_ConfigureInfo for information on the format of this list). If option is specified with no value, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no option is specified). If one or more option−value pairs are specified, then the command modifies the given option(s) to have the given value(s); in this case the command returns an empty string. See EMBEDDED WINDOWS for information on the options that are supported.

pathName window create index ?option value ...?
This command creates a new window annotation, which will appear in the text at the position given by index. Any number of option−value pairs may be specified to configure the annotation. See EMBEDDED WINDOWS for information on the options that are supported. Returns an empty string.

pathName window names
Returns a list whose elements are the names of all windows currently embedded in window.

pathName xview option args
This command is used to query and change the horizontal position of the text in the widget's window. It can take any of the following forms:

pathName xview
Returns a list containing two elements. Each element is a real fraction between 0 and 1; together they describe the portion of the document's horizontal span that is visible in the window. For example, if the first element is .2 and the second element is .6, 20% of the text is off−screen to the left, the middle 40% is visible in the window, and 40% of the text is off−screen to the right. The fractions refer only to the lines that are actually visible in the window: if the lines in the window are all very short, so that they are entirely visible, the returned fractions will be 0 and 1, even if there are other lines in the text that are much wider than the window. These are the same values passed to scrollbars via the –xscrollcommand option.

pathName xview moveto fraction
Adjusts the view in the window so that fraction of the horizontal span of the text is off−screen to the left. Fraction is a fraction between 0 and 1.

pathName xview scroll number what
This command shifts the view in the window left or right according to number and what. Number must be an integer. What must be either units or pages or an abbreviation of one of
these. If *what* is **units**, the view adjusts left or right by *number* average−width characters on the display; if it is **pages** then the view adjusts by *number* screenfuls. If *number* is negative then characters farther to the left become visible; if it is positive then characters farther to the right become visible.

`pathName yview ?args?`

This command is used to query and change the vertical position of the text in the widget's window. It can take any of the following forms:

`pathName yview`

Returns a list containing two elements, both of which are real fractions between 0 and 1. The first element gives the position of the first character in the top line in the window, relative to the text as a whole (0.5 means it is halfway through the text, for example). The second element gives the position of the character just after the last one in the bottom line of the window, relative to the text as a whole. These are the same values passed to scrollbars via the −yscrollcommand option.

`pathName yview moveto fraction`

Adjusts the view in the window so that the character given by *fraction* appears on the top line of the window. *Fraction* is a fraction between 0 and 1; 0 indicates the first character in the text, 0.33 indicates the character one−third the way through the text, and so on.

`pathName yview scroll number what`

This command adjust the view in the window up or down according to *number* and *what*. *Number* must be an integer. *What* must be either **units** or **pages**. If *what* is **units**, the view adjusts up or down by *number* lines on the display; if it is **pages** then the view adjusts by *number* screenfuls. If *number* is negative then earlier positions in the text become visible; if it is positive then later positions in the text become visible.

`pathName yview ?−pickplace? index`

Changes the view in the widget's window to make *index* visible. If the −pickplace option isn't specified then *index* will appear at the top of the window. If −pickplace is specified then the widget chooses where *index* appears in the window:

[1] If *index* is already visible somewhere in the window then the command does nothing.

[2] If *index* is only a few lines off−screen above the window then it will be positioned at the top of the window.

[3] If *index* is only a few lines off−screen below the window then it will be positioned at the bottom of the window.

[4] Otherwise, *index* will be centered in the window.

The −pickplace option has been obsoleted by the see widget command (see handles both x− and y−motion to make a location visible, whereas −pickplace only handles motion in y).

`pathName yview number`

This command makes the first character on the line after the one given by *number* visible at the top of the window. *Number* must be an integer. This command used to be used for
scrolling, but now it is obsolete.

BINDINGS

Tk automatically creates class bindings for texts that give them the following default behavior. In the descriptions below, "word" is dependent on the value of the `tcl_wordchars` variable. See `tclvars(n)`.

[1] Clicking mouse button 1 positions the insertion cursor just before the character underneath the mouse cursor, sets the input focus to this widget, and clears any selection in the widget. Dragging with mouse button 1 strokes out a selection between the insertion cursor and the character under the mouse.

[2] Double-clicking with mouse button 1 selects the word under the mouse and positions the insertion cursor at the end of the word. Dragging after a double click will stroke out a selection consisting of whole words.

[3] Triple-clicking with mouse button 1 selects the line under the mouse and positions the insertion cursor at the end of the line. Dragging after a triple click will stroke out a selection consisting of whole lines.

[4] The ends of the selection can be adjusted by dragging with mouse button 1 while the Shift key is down; this will adjust the end of the selection that was nearest to the mouse cursor when button 1 was pressed. If the button is double-clicked before dragging then the selection will be adjusted in units of whole words; if it is triple-clicked then the selection will be adjusted in units of whole lines.

[5] Clicking mouse button 1 with the Control key down will reposition the insertion cursor without affecting the selection.

[6] If any normal printing characters are typed, they are inserted at the point of the insertion cursor.

[7] The view in the widget can be adjusted by dragging with mouse button 2. If mouse button 2 is clicked without moving the mouse, the selection is copied into the text at the position of the mouse cursor. The Insert key also inserts the selection, but at the position of the insertion cursor.

[8] If the mouse is dragged out of the widget while button 1 is pressed, the entry will automatically scroll to make more text visible (if there is more text off-screen on the side where the mouse left the window).

[9] The Left and Right keys move the insertion cursor one character to the left or right; they also clear any selection in the text. If Left or Right is typed with the Shift key down, then the insertion cursor moves and the selection is extended to include the new character. Control–Left and Control–Right move the insertion cursor by words, and Control–Shift–Left and Control–Shift–Right move the insertion cursor by words and also extend the selection. Control–b and Control–f behave the same as Left and Right, respectively. Meta–b and Meta–f behave the same as Control–Left and
Control–Right, respectively.

10 The Up and Down keys move the insertion cursor one line up or down and clear any selection in the text. If Up or Right is typed with the Shift key down, then the insertion cursor moves and the selection is extended to include the new character. Control–Up and Control–Down move the insertion cursor by paragraphs (groups of lines separated by blank lines), and Control–Shift–Up and Control–Shift–Down move the insertion cursor by paragraphs and also extend the selection. Control–p and Control–n behave the same as Up and Down, respectively.

11 The Next and Prior keys move the insertion cursor forward or backwards by one screenful and clear any selection in the text. If the Shift key is held down while Next or Prior is typed, then the selection is extended to include the new character. Control–v moves the view down one screenful without moving the insertion cursor or adjusting the selection.

12 Control–Next and Control–Prior scroll the view right or left by one page without moving the insertion cursor or affecting the selection.

13 Home and Control–a move the insertion cursor to the beginning of its line and clear any selection in the widget. Shift–Home moves the insertion cursor to the beginning of the line and also extends the selection to that point.

14 End and Control–e move the insertion cursor to the end of the line and clear any selection in the widget. Shift–End moves the cursor to the end of the line and extends the selection to that point.

15 Control–Home and Meta–< move the insertion cursor to the beginning of the text and clear any selection in the widget. Control–Shift–Home moves the insertion cursor to the beginning of the text and also extends the selection to that point.

16 Control–End and Meta–> move the insertion cursor to the end of the text and clear any selection in the widget. Control–Shift–End moves the cursor to the end of the text and extends the selection to that point.

17 The Select key and Control–Space set the selection anchor to the position of the insertion cursor. They don't affect the current selection. Shift–Select and Control–Shift–Space adjust the selection to the current position of the insertion cursor, selecting from the anchor to the insertion cursor if there was not any selection previously.

18 Control–/ selects the entire contents of the widget.

19 Control–\ clears any selection in the widget.

20 The F16 key (labelled Copy on many Sun workstations) or Meta–w copies the selection in the widget to the clipboard, if there is a selection. This action is carried out by the command $tk_textCopy$.

21 The F20 key (labelled Cut on many Sun workstations) or Control–w copies the selection in the widget
to the clipboard and deletes the selection. This action is carried out by the command \texttt{tk\_textCut}. If there is no selection in the widget then these keys have no effect.

The F18 key (labelled Paste on many Sun workstations) or Control–y inserts the contents of the clipboard at the position of the insertion cursor. This action is carried out by the command \texttt{tk\_textPaste}.

The Delete key deletes the selection, if there is one in the widget. If there is no selection, it deletes the character to the right of the insertion cursor.

Backspace and Control–h delete the selection, if there is one in the widget. If there is no selection, they delete the character to the left of the insertion cursor.

Control–d deletes the character to the right of the insertion cursor.

Meta–d deletes the word to the right of the insertion cursor.

Control–k deletes from the insertion cursor to the end of its line; if the insertion cursor is already at the end of a line, then Control–k deletes the newline character.

Control–o opens a new line by inserting a newline character in front of the insertion cursor without moving the insertion cursor.

Meta–backspace and Meta–Delete delete the word to the left of the insertion cursor.

Control–x deletes whatever is selected in the text widget after copying it to the clipboard.

Control–t reverses the order of the two characters to the right of the insertion cursor.

Control–z (and Control–underscore on UNIX when \texttt{tk\_strictMotif} is true) undoes the last edit action if the \texttt{–undo} option is true. Does nothing otherwise.

Control–Z (or Control–y on Windows) reapplies the last undone edit action if the \texttt{–undo} option is true. Does nothing otherwise.

If the widget is disabled using the \texttt{–state} option, then its view can still be adjusted and text can still be selected, but no insertion cursor will be displayed and no text modifications will take place.

The behavior of texts can be changed by defining new bindings for individual widgets or by redefining the class bindings.

**PERFORMANCE ISSUES**

Text widgets should run efficiently under a variety of conditions. The text widget uses about 2–3 bytes of main memory for each byte of text, so texts containing a megabyte or more should be practical on most
workstations. Text is represented internally with a modified B−tree structure that makes operations relatively efficient even with large texts. Tags are included in the B−tree structure in a way that allows tags to span large ranges or have many disjoint smaller ranges without loss of efficiency. Marks are also implemented in a way that allows large numbers of marks. In most cases it is fine to have large numbers of unique tags, or a tag that has many distinct ranges.

One performance problem can arise if you have hundreds or thousands of different tags that all have the following characteristics: the first and last ranges of each tag are near the beginning and end of the text, respectively, or a single tag range covers most of the text widget. The cost of adding and deleting tags like this is proportional to the number of other tags with the same properties. In contrast, there is no problem with having thousands of distinct tags if their overall ranges are localized and spread uniformly throughout the text.

Very long text lines can be expensive, especially if they have many marks and tags within them.

The display line with the insert cursor is redrawn each time the cursor blinks, which causes a steady stream of graphics traffic. Set the insertOffTime attribute to 0 to avoid this.

SEE ALSO

entry, scrollbar

KEYWORDS

text, widget, tkvars

Copyright ' 1992 The Regents of the University of California.
Copyright ' 1994−1996 Sun Microsystems, Inc.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
tk

NAME

tk – Manipulate Tk internal state

SYNOPSIS

```
tk option ?arg arg ...?
```

DESCRIPTION

The `tk` command provides access to miscellaneous elements of Tk's internal state. Most of the information manipulated by this command pertains to the application as a whole, or to a screen or display, rather than to a particular window. The command can take any of a number of different forms depending on the `option` argument. The legal forms are:

```
tk appname ?newName?
```

If `newName` isn't specified, this command returns the name of the application (the name that may be used in `send` commands to communicate with the application). If `newName` is specified, then the name of the application is changed to `newName`. If the given name is already in use, then a suffix of the form ```#2``` or ```#3``` is appended in order to make the name unique. The command's result is the name actually chosen. `newName` should not start with a capital letter. This will interfere with option processing, since names starting with capitals are assumed to be classes; as a result, Tk may not be able to find some options for the application. If sends have been disabled by deleting the `send` command, this command will reenable them and recreate the `send` command.

```
tk caret window ?–x x? ?–y y? ?–height height?
```

Sets and queries the caret location for the display of the specified Tk window `window`. The caret is the per–display cursor location used for indicating global focus (e.g. to comply with Microsoft Accessibility guidelines), as well as for location of the over–the–spot XIM (X Input Methods) or

```
tk scaling ?–displayof window? ?number?
```

```
tk useinputmethods ?–displayof window? ?boolean?
```

```
tk windowingsystem
```

KEYWORDS

NAME

tk – Manipulate Tk internal state

SYNOPSIS

```
tk option ?arg arg ...?
```

DESCRIPTION

The `tk` command provides access to miscellaneous elements of Tk's internal state. Most of the information manipulated by this command pertains to the application as a whole, or to a screen or display, rather than to a particular window. The command can take any of a number of different forms depending on the `option` argument. The legal forms are:

```
tk appname ?newName?
```

If `newName` isn't specified, this command returns the name of the application (the name that may be used in `send` commands to communicate with the application). If `newName` is specified, then the name of the application is changed to `newName`. If the given name is already in use, then a suffix of the form ```#2``` or ```#3``` is appended in order to make the name unique. The command's result is the name actually chosen. `newName` should not start with a capital letter. This will interfere with option processing, since names starting with capitals are assumed to be classes; as a result, Tk may not be able to find some options for the application. If sends have been disabled by deleting the `send` command, this command will reenable them and recreate the `send` command.

```
tk caret window ?–x x? ?–y y? ?–height height?
```

Sets and queries the caret location for the display of the specified Tk window `window`. The caret is the per–display cursor location used for indicating global focus (e.g. to comply with Microsoft Accessibility guidelines), as well as for location of the over–the–spot XIM (X Input Methods) or
Windows IME windows. If no options are specified, the last values used for setting the caret are return in option–value pair format. \(−x\) and \(−y\) represent window–relative coordinates, and \(−\text{height}\) is the height of the current cursor location, or the height of the specified \textit{window} if none is given.

\textbf{tk scaling} \textit{?–displayof} \textit{window} \textit{?number}?

Sets and queries the current scaling factor used by Tk to convert between physical units (for example, points, inches, or millimeters) and pixels. The \textit{number} argument is a floating point number that specifies the number of pixels per point on \textit{window}'s display. If the \textit{window} argument is omitted, it defaults to the main window. If the \textit{number} argument is omitted, the current value of the scaling factor is returned.

A ``point'' is a unit of measurement equal to 1/72 inch. A scaling factor of 1.0 corresponds to 1 pixel per point, which is equivalent to a standard 72 dpi monitor. A scaling factor of 1.25 would mean 1.25 pixels per point, which is the setting for a 90 dpi monitor; setting the scaling factor to 1.25 on a 72 dpi monitor would cause everything in the application to be displayed 1.25 times as large as normal. The initial value for the scaling factor is set when the application starts, based on properties of the installed monitor, but it can be changed at any time. Measurements made after the scaling factor is changed will use the new scaling factor, but it is undefined whether existing widgets will resize themselves dynamically to accommodate the new scaling factor.

\textbf{tk useinputmethods} \textit{?–displayof} \textit{window} \textit{?boolean}?

Sets and queries the state of whether Tk should use XIM (X Input Methods) for filtering events. The resulting state is returned. XIM is used in some locales (ie: Japanese, Korean), to handle special input devices. This feature is only significant on X. If XIM support is not available, this will always return 0. If the \textit{window} argument is omitted, it defaults to the main window. If the \textit{boolean} argument is omitted, the current state is returned. This is turned on by default for the main display.

\textbf{tk windowingsystem}

Returns the current Tk windowing system, one of \textbf{x11} (X11–based), \textbf{win32} (MS Windows), \textbf{classic} (Mac OS Classic), or \textbf{aqua} (Mac OS X Aqua).

\textbf{KEYWORDS}

\textit{application name}, \texttt{send}

\textit{Copyright} ' 1992 The Regents of the University of California.
\textit{Copyright} ' 1994–1996 Sun Microsystems, Inc.
chooseColor

NAME

tk_chooseColor – pops up a dialog box for the user to select a color.

SYNOPSIS

tk_chooseColor ?option value ...?

DESCRIPTION

The procedure tk_chooseColor pops up a dialog box for the user to select a color. The following option−value pairs are possible as command line arguments:

−initialcolor color
   Specifies the color to display in the color dialog when it pops up. color must be in a form acceptable to the Tk_GetColor function.

−parent window
   Makes window the logical parent of the color dialog. The color dialog is displayed on top of its parent window.

−title titleString
   Specifies a string to display as the title of the dialog box. If this option is not specified, then a default title will be displayed.

If the user selects a color, tk_chooseColor will return the name of the color in a form acceptable to Tk_GetColor. If the user cancels the operation, both commands will return the empty string.

EXAMPLE

button .b −bg [tk_chooseColor −initialcolor gray −title "Choose color"]

KEYWORDS

color selection dialog

Copyright ' 1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
chooseDirectory

NAME

tk_chooseDirectory – pops up a dialog box for the user to select a directory.

SYNOPSIS

    tk_chooseDirectory ?option value ...?

DESCRIPTION

    −initialdir dirname
    −parent window
    −title titleString
    −mustexist boolean

EXAMPLE

SEE ALSO

KEYWORDS

NAME

tk_chooseDirectory – pops up a dialog box for the user to select a directory.

SYNOPSIS

    tk_chooseDirectory ?option value ...?

DESCRIPTION

The procedure tk_chooseDirectory pops up a dialog box for the user to select a directory. The following option−value pairs are possible as command line arguments:

    −initialdir dirname
        Specifies that the directories in directory should be displayed when the dialog pops up. If this parameter is not specified, then the directories in the current working directory are displayed. If the parameter specifies a relative path, the return value will convert the relative path to an absolute path. This option may not always work on the Macintosh. This is not a bug. Rather, the General Controls control panel on the Mac allows the end user to override the application default directory.

    −parent window
        Makes window the logical parent of the dialog. The dialog is displayed on top of its parent window.

    −title titleString
        Specifies a string to display as the title of the dialog box. If this option is not specified, then a default title will be displayed.

    −mustexist boolean
        Specifies whether the user may specify non−existent directories. If this parameter is true, then the user may only select directories that already exist. The default value is false.
EXAMPLE

    set dir [tk_chooseDirectory \    -initialdir ~ -title "Choose a directory"]
    if {$dir eq ""} {
        label .l -text "No directory selected"
    } else {
        label .l -text "Selected $dir"
    }

SEE ALSO

    tk_getOpenFile, tk_getSaveFile

KEYWORDS

    directory, selection, dialog, platform−specific
NAME

tk_dialog – Create modal dialog and wait for response

SYNOPSIS

**tk_dialog** window title text bitmap default string string ...

DESCRIPTION

**window**
Name of top-level window to use for dialog. Any existing window by this name is destroyed.

**title**
Text to appear in the window manager's title bar for the dialog.

**text**
Message to appear in the top portion of the dialog box.

**bitmap**
If non-empty, specifies a bitmap to display in the top portion of the dialog, to the left of the text. If this is an empty string then no bitmap is displayed in the dialog.

**default**
If this is an integer greater than or equal to zero, then it gives the index of the button that is to be the default button for the dialog (0 for the leftmost button, and so on). If less than zero or an empty string then there won't be any default button.
There will be one button for each of these arguments. Each string specifies text to display in a button, in order from left to right.

After creating a dialog box, \texttt{tk\_dialog} waits for the user to select one of the buttons either by clicking on the button with the mouse or by typing return to invoke the default button (if any). Then it returns the index of the selected button: 0 for the leftmost button, 1 for the button next to it, and so on. If the dialog's window is destroyed before the user selects one of the buttons, then \texttt{-1} is returned.

While waiting for the user to respond, \texttt{tk\_dialog} sets a local grab. This prevents the user from interacting with the application in any way except to invoke the dialog box.

\textbf{EXAMPLE}

\begin{verbatim}
set reply [tk\_dialog .foo "The Title" "Do you want to say yes?" questhead 0 Yes No "I'm not sure"]
\end{verbatim}

\textbf{SEE ALSO}

\texttt{tk\_messageBox}

\textbf{KEYWORDS}

\texttt{bitmap, dialog, modal}

\texttt{Copyright ' 1992 The Regents of the University of California.}
\texttt{Copyright ' 1994-1996 Sun Microsystems, Inc.}
\texttt{Copyright ' 1995-1997 Roger E. Critchlow Jr.}
**focusNext**

**NAME**

tk_focusNext, tk_focusPrev, tk_focusFollowsMouse – Utility procedures for managing the input focus.

**SYNOPSIS**

```tcl
tk_focusNext window
tk_focusPrev window
tk_focusFollowsMouse
```

**DESCRIPTION**

**tk_focusNext** is a utility procedure used for keyboard traversal. It returns the ``next'' window after `window` in focus order. The focus order is determined by the stacking order of windows and the structure of the window hierarchy. Among siblings, the focus order is the same as the stacking order, with the lowest window being first. If a window has children, the window is visited first, followed by its children (recursively), followed by its next sibling. Top−level windows other than `window` are skipped, so that **tk_focusNext** never returns a window in a different top−level from `window`.

After computing the next window, **tk_focusNext** examines the window's −takefocus option to see whether it should be skipped. If so, **tk_focusNext** continues on to the next window in the focus order, until it eventually finds a window that will accept the focus or returns back to `window`.

**tk_focusPrev** is similar to **tk_focusNext** except that it returns the window just before `window` in the focus order.

**tk_focusFollowsMouse** changes the focus model for the application to an implicit one where the window under the mouse gets the focus. After this procedure is called, whenever the mouse enters a window Tk will automatically give it the input focus. The focus command may be used to move the focus to a window other than the one under the mouse, but as soon as the mouse moves into a new window the focus will jump to that window. Note: at present there is no built−in support for returning the application to an explicit focus model; to do this you'll have to write a script that deletes the bindings created by **tk_focusFollowsMouse**.

**KEYWORDS**

focus, keyboard traversal, top−level

---

Copyright ' 1994 The Regents of the University of California.  
Copyright ' 1994−1996 Sun Microsystems, Inc.  
Copyright ' 1995−1997 Roger E. Critchlow Jr.
NAME

tk_getOpenFile, tk_getSaveFile – pop up a dialog box for the user to select a file to open or save.

SYNOPSIS

    tk_getOpenFile ?option value ...?
    tk_getSaveFile ?option value ...?

DESCRIPTION

SPECIFYING FILE PATTERNS
SPECIFYING EXTENSIONS
EXAMPLE
SEE ALSO
KEYWORDS

### NAME

tk_getOpenFile, tk_getSaveFile – pop up a dialog box for the user to select a file to open or save.

### SYNOPSIS

```
    tk_getOpenFile ?option value ...?
    tk_getSaveFile ?option value ...?
```

### DESCRIPTION

The procedures `tk_getOpenFile` and `tk_getSaveFile` pop up a dialog box for the user to select a file to open or save. The `tk_getOpenFile` command is usually associated with the `Open` command in the `File` menu. Its purpose is for the user to select an existing file _only_. If the user enters a non-existent file, the dialog box gives the user an error prompt and requires the user to give an alternative selection. If an application allows the user to create new files, it should do so by providing a separate `New` menu command.

The `tk_getSaveFile` command is usually associated with the `Save as` command in the `File` menu. If the user enters a file that already exists, the dialog box prompts the user for confirmation whether the existing file
should be overwritten or not.

The following option–value pairs are possible as command line arguments to these two commands:

- `--defaultextension extension`
  Specifies a string that will be appended to the filename if the user enters a filename without an extension. The default value is the empty string, which means no extension will be appended to the filename in any case. This option is ignored on the Macintosh platform, which does not require extensions to filenames, and the UNIX implementation guesses reasonable values for this from the `--filetypes` option when this is not supplied.

- `--filetypes filePatternList`
  If a File types listbox exists in the file dialog on the particular platform, this option gives the filetypes in this listbox. When the user choose a filetype in the listbox, only the files of that type are listed. If this option is unspecified, or if it is set to the empty list, or if the File types listbox is not supported by the particular platform then all files are listed regardless of their types. See the section SPECIFYING FILE PATTERNS below for a discussion on the contents of filePatternList.

- `--initialdir directory`
  Specifies that the files in directory should be displayed when the dialog pops up. If this parameter is not specified, then the files in the current working directory are displayed. If the parameter specifies a relative path, the return value will convert the relative path to an absolute path. This option may not always work on the Macintosh. This is not a bug. Rather, the General Controls control panel on the Mac allows the end user to override the application default directory.

- `--initialfile filename`
  Specifies a filename to be displayed in the dialog when it pops up. This option is ignored on the Macintosh platform.

- `--multiple`
  Allows the user to choose multiple files from the Open dialog. On the Macintosh, this is only available when Navigation Services are installed.

- `--message string`
  Specifies a message to include in the client area of the dialog. This is only available on the Macintosh, and only when Navigation Services are installed.

- `--parent window`
  Makes window the logical parent of the file dialog. The file dialog is displayed on top of its parent window.

- `--title titleString`
  Specifies a string to display as the title of the dialog box. If this option is not specified, then a default title is displayed.

If the user selects a file, both `tk_getOpenFile` and `tk_getSaveFile` return the full pathname of this file. If the user cancels the operation, both commands return the empty string.

**SPECIFYING FILE PATTERNS**

The filePatternList value given by the `--filetypes` option is a list of file patterns. Each file pattern is a list of the form
typeName (extension ?extension ...?) ?{macType ?macType ...?}?  

typeName is the name of the file type described by this file pattern and is the text string that appears in the File types listbox. extension is a file extension for this file pattern. macType is a four–character Macintosh file type. The list of macTypes is optional and may be omitted for applications that do not need to execute on the Macintosh platform.

Several file patterns may have the same typeName, in which case they refer to the same file type and share the same entry in the listbox. When the user selects an entry in the listbox, all the files that match at least one of the file patterns corresponding to that entry are listed. Usually, each file pattern corresponds to a distinct type of file. The use of more than one file patterns for one type of file is necessary on the Macintosh platform only.

On the Macintosh platform, a file matches a file pattern if its name matches at least one of the extension(s) AND it belongs to at least one of the macType(s) of the file pattern. For example, the C Source Files file pattern in the sample code matches with files that have a .c extension AND belong to the macType TEXT. To use the OR rule instead, you can use two file patterns, one with the extensions only and the other with the macType only. The GIF Files file type in the sample code matches files that EITHER have a .gif extension OR belong to the macType GIFF.

On the Unix and Windows platforms, a file matches a file pattern if its name matches at least one of the extension(s) of the file pattern. The macTypes are ignored.

**SPECIFYING EXTENSIONS**

On the Unix and Macintosh platforms, extensions are matched using glob–style pattern matching. On the Windows platforms, extensions are matched by the underlying operating system. The types of possible extensions are: (1) the special extension * matches any file; (2) the special extension "" matches any files that do not have an extension (i.e., the filename contains no full stop character); (3) any character string that does not contain any wild card characters (* and ?).

Due to the different pattern matching rules on the various platforms, to ensure portability, wild card characters are not allowed in the extensions, except as in the special extension *. Extensions without a full stop character (e.g. ~) are allowed but may not work on all platforms.

**EXAMPLE**

```tcl
set types {
    {Text Files} {.txt}  
    {TCL Scripts} {.tcl}  
    {C Source Files} {.c} TEXT  
    {GIF Files} {.gif}  
    {GIF Files} {} GIFF  
    {All Files} *  
}
set filename [tk_getOpenFile -filetypes $types]
if {$filename != ""} {
```
# Open the file ...
}

SEE ALSO

tk_chooseDirectory

KEYWORDS

file selection dialog

Copyright © 1996 Sun Microsystems, Inc.
Copyright © 1995-1997 Roger E. Critchlow Jr.
messageBox

NAME

tk_messageBox – pops up a message window and waits for user response.

SYNOPSIS

tk_messageBox ?option value ...?

DESCRIPTION

−default name
−icon iconImage
−message string
−parent window
−title titleString
−type predefinedType
  abortretryignore
  ok
  okcancel
  retrycancel
  yesno
  yesnocancel

EXAMPLE

KEYWORDS

NAME

tk_messageBox – pops up a message window and waits for user response.

SYNOPSIS

tk_messageBox ?option value ...?

DESCRIPTION

This procedure creates and displays a message window with an application−specified message, an icon and a set of buttons. Each of the buttons in the message window is identified by a unique symbolic name (see the −type options). After the message window is popped up, tk_messageBox waits for the user to select one of the buttons. Then it returns the symbolic name of the selected button. The following option−value pairs are supported:

−default name

Name gives the symbolic name of the default button for this message window ('ok', 'cancel', and so on). See −type for a list of the symbolic names. If this option is not specified, the first button in the dialog will be made the default.
−icon iconImage
   Specifies an icon to display. IconImage must be one of the following: error, info, question or
   warning. If this option is not specified, then the info icon will be displayed.

−message string
   Specifies the message to display in this message box.

−parent window
   Makes window the logical parent of the message box. The message box is displayed on top of its
   parent window.

−title titleString
   Specifies a string to display as the title of the message box. The default value is an empty string.

−type predefinedType
   Arranges for a predefined set of buttons to be displayed. The following values are possible for
   predefinedType:

   abortretryignore
      Displays three buttons whose symbolic names are abort, retry and ignore.

   ok
      Displays one button whose symbolic name is ok.

   okcancel
      Displays two buttons whose symbolic names are ok and cancel.

   retrycancel
      Displays two buttons whose symbolic names are retry and cancel.

   yesno
      Displays two buttons whose symbolic names are yes and no.

   yesnocancel
      Displays three buttons whose symbolic names are yes, no and cancel.

EXAMPLE

set answer [tk_messageBox -message "Really quit?" -type yesno -icon question]
switch -- $answer {
   yes exit
   no {tk_messageBox -message "I know you like this application!" \
      -type ok}
}

KEYWORDS

message box

Copyright ' 1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
optionMenu

NAME

tk_optionMenu – Create an option menubutton and its menu

SYNOPSIS

tk_optionMenu w varName value ?value value ...?

DESCRIPTION

This procedure creates an option menubutton whose name is \textit{w}, plus an associated menu. Together they allow the user to select one of the values given by the \textit{value} arguments. The current value will be stored in the global variable whose name is given by \textit{varName} and it will also be displayed as the label in the option menubutton. The user can click on the menubutton to display a menu containing all of the \textit{values} and thereby select a new value. Once a new value is selected, it will be stored in the variable and appear in the option menubutton. The current value can also be changed by setting the variable.

The return value from \texttt{tk_optionMenu} is the name of the menu associated with \textit{w}, so that the caller can change its configuration options or manipulate it in other ways.

KEYWORDS

\texttt{option menu}

\texttt{Copyright ' 1990-1994 The Regents of the University of California.}
\texttt{Copyright ' 1994-1996 Sun Microsystems, Inc.}
\texttt{Copyright ' 1995-1997 Roger E. Critchlow Jr.}
popup

NAME

tk_popup – Post a popup menu

SYNOPSIS

tk_popup menu x y ?entry?

DESCRIPTION

This procedure posts a menu at a given position on the screen and configures Tk so that the menu and its cascaded children can be traversed with the mouse or the keyboard. Menu is the name of a menu widget and x and y are the root coordinates at which to display the menu. If entry is omitted or an empty string, the menu's upper left corner is positioned at the given point. Otherwise entry gives the index of an entry in menu and the menu will be positioned so that the entry is positioned over the given point.

EXAMPLE

How to attach a simple popup menu to a widget.

    # Create a menu
    set m [menu .popupMenu]
    $m add command -label "Example 1" -command bell
    $m add command -label "Example 2" -command bell

    # Create something to attach it to
    pack [label .l -text "Click me!"]

    # Arrange for the menu to pop up when the label is clicked
    bind .l <1> {tk_popup .popupMenu %X %Y}

SEE ALSO

    bind, menu, tk_optionMenu

KEYWORDS

    menu, popup
palette

NAME

tk_setPalette, tk_bisque – Modify the Tk color palette

SYNOPSIS

tk_setPalette background
tk_setPalette name value ?name value ...?
tk_bisque

DESCRIPTION

The tk_setPalette procedure changes the color scheme for Tk. It does this by modifying the colors of existing widgets and by changing the option database so that future widgets will use the new color scheme. If tk_setPalette is invoked with a single argument, the argument is the name of a color to use as the normal background color; tk_setPalette will compute a complete color palette from this background color. Alternatively, the arguments to tk_setPalette may consist of any number of name-value pairs, where the first argument of the pair is the name of an option in the Tk option database and the second argument is the new value to use for that option. The following database names are currently supported:

activeBackground     foreground     selectColor
activeForeground     highlightBackground     selectBackground
background     highlightColor  selectForeground
disabledForeground     insertBackground        troughColor

tk_setPalette tries to compute reasonable defaults for any options that you don't specify. You can specify options other than the above ones and Tk will change those options on widgets as well. This feature may be useful if you are using custom widgets with additional color options.

Once it has computed the new value to use for each of the color options, tk_setPalette scans the widget hierarchy to modify the options of all existing widgets. For each widget, it checks to see if any of the above options is defined for the widget. If so, and if the option's current value is the default, then the value is changed; if the option has a value other than the default, tk_setPalette will not change it. The default for an option is the one provided by the widget ([lindex [sw configure $option] 3]) unless tk_setPalette has been run previously, in which case it is the value specified in the previous invocation of tk_setPalette.

After modifying all the widgets in the application, tk_setPalette adds options to the option database to change the defaults for widgets created in the future. The new options are added at priority widgetDefault, so they will be overridden by options from the .Xdefaults file or options specified on the command-line that creates a widget.

The procedure tk_bisque is provided for backward compatibility: it restores the application's colors to the light brown ("bisque") color scheme used in Tk 3.6 and earlier versions.
tkerror

NAME

tkerror — Command invoked to process background errors

SYNOPSIS

tkerror message

DESCRIPTION

Note: as of Tk 4.1 the tkerror command has been renamed to bgerror because the event loop (which is what usually invokes it) is now part of Tcl. For backward compatibility the bgerror provided by the current Tk version still tries to call tkerror if there is one (or an auto loadable one), so old script defining that error handler should still work, but you should anyhow modify your scripts to use bgerror instead of tkerror because that support for the old name might vanish in the near future. If that call fails, bgerror posts a dialog showing the error and offering to see the stack trace to the user. If you want your own error management you should directly override bgerror instead of tkerror. Documentation for bgerror is available as part of Tcl's documentation.

KEYWORDS

background error, reporting

Copyright © 1990−1994 The Regents of the University of California.
Copyright © 1994−1996 Sun Microsystems, Inc.
Copyright © 1995−1997 Roger E. Critchlow Jr.
tkvars

NAME

tkvars – Variables used or set by Tk

DESCRIPTION

The following Tcl variables are either set or used by Tk at various times in its execution:

*tk_library*
This variable holds the file name for a directory containing a library of Tcl scripts related to Tk. These scripts include an initialization file that is normally processed whenever a Tk application starts up, plus other files containing procedures that implement default behaviors for widgets. The initial value of `tcl_library` is set when Tk is added to an interpreter; this is done by searching several different directories until one is found that contains an appropriate Tk startup script. If the `TK_LIBRARY` environment variable exists, then the directory it names is checked first. If `TK_LIBRARY` isn't set or doesn't refer to an appropriate directory, then Tk checks several other directories based on a compiled-in default location, the location of the Tcl library directory, the location of the binary containing the application, and the current working directory. The variable can be modified by an application to switch to a different library.

*tk_patchLevel*
Contains a decimal integer giving the current patch level for Tk. The patch level is incremented for each new release or patch, and it uniquely identifies an official version of Tk.

*tk::Priv*
This variable is an array containing several pieces of information that are private to Tk. The elements of `tk::Priv` are used by Tk library procedures and default bindings. They should not be accessed by any code outside Tk.

*tk_strictMotif*
This variable is set to zero by default. If an application sets it to one, then Tk attempts to adhere as closely as possible to Motif look-and-feel standards. For example, active elements such as buttons and scrollbar sliders will not change color when the pointer passes over them.

*tk_textRedraw*
*tk_textRelayout*
These variables are set by text widgets when they have debugging turned on. The values written to these variables can be used to test or debug text widget operations. These variables are mostly used by Tk's test suite.

*tk_version*
Tk sets this variable in the interpreter for each application. The variable holds the current version number of the Tk library in the form `major.minor`. `Major` and `minor` are integers. The major version number increases in any Tk release that includes changes that are not backward compatible (i.e. whenever existing Tk applications and scripts may have to change to work with the new release). The minor version number increases with each new release of Tk, except that it resets to zero whenever the major version number changes.
tkwait

NAME

tkwait – Wait for variable to change or window to be destroyed

SYNOPSIS

    tkwait variable name
    tkwait visibility name
    tkwait window name

DESCRIPTION

The tkwait command waits for one of several things to happen, then it returns without taking any other actions. The return value is always an empty string. If the first argument is variable (or any abbreviation of it) then the second argument is the name of a global variable and the command waits for that variable to be modified. If the first argument is visibility (or any abbreviation of it) then the second argument is the name of a window and the tkwait command waits for a change in its visibility state (as indicated by the arrival of a VisibilityNotify event). This form is typically used to wait for a newly-created window to appear on the screen before taking some action. If the first argument is window (or any abbreviation of it) then the second argument is the name of a window and the tkwait command waits for that window to be destroyed. This form is typically used to wait for a user to finish interacting with a dialog box before using the result of that interaction.

While the tkwait command is waiting it processes events in the normal fashion, so the application will continue to respond to user interactions. If an event handler invokes tkwait again, the nested call to tkwait must complete before the outer call can complete.

KEYWORDS

variable, visibility, wait, window

Copyright ' 1992 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
toplevel

NAME
toplevel – Create and manipulate toplevel widgets

SYNOPSIS
toplevel pathName ?options?

STANDARD OPTIONS
−borderwidth or −bd, borderWidth, BorderWidth
−cursor, cursor, Cursor
−highlightbackground, highlightBackground, HighlightBackground
−highlightcolor, highlightColor, HighlightColor
−highlightthickness, highlightThickness, HighlightThickness
−padX, padX, Pad
−padY, padY, Pad
−relief, relief, Relief
−takefocus, takeFocus, TakeFocus

WIDGET−SPECIFIC OPTIONS
−background, background, Background
−class, class, Class
−colormap, colormap, Colormap
−container, container, Container
−height, height, Height
−menu, menu, Menu
−screen,
−use, use, Use
−visual, visual, Visual
−width, width, Width

DESCRIPTION

WIDGET COMMAND
pathName cget option
pathName configure ?option? ?value option value ...?

BINDINGS

SEE ALSO

KEYWORDS
SYNOPSIS

toplevel pathName ?options?

STANDARD OPTIONS

−borderwidth or −bd, borderWidth, BorderWidth
−cursor, cursor, Cursor
−highlightbackground, highlightBackground, HighlightBackground
−highlightcolor, highlightColor, HighlightColor
−highlightthickness, highlightThickness, HighlightThickness
−padx, padX, Pad
−pady, padY, Pad
−relief, relief, Relief
−takefocus, takeFocus, TakeFocus

WIDGET−SPECIFIC OPTIONS

Command−Line Name: −background
Database Name: background
Database Class: Background

This option is the same as the standard background option except that its value may also be specified
as an empty string. In this case, the widget will display no background or border, and no colors will be
consumed from its colormap for its background and border.

Command−Line Name: −class
Database Name: class
Database Class: Class

Specifies a class for the window. This class will be used when querying the option database for the
window's other options, and it will also be used later for other purposes such as bindings. The class
option may not be changed with the configure widget command.

Command−Line Name: −colormap
Database Name: colormap
Database Class: Colormap

Specifies a colormap to use for the window. The value may be either new, in which case a new
colormap is created for the window and its children, or the name of another window (which must be
on the same screen and have the same visual as pathName), in which case the new window will use
the colormap from the specified window. If the colormap option is not specified, the new window
uses the default colormap of its screen. This option may not be changed with the configure widget
command.

Command−Line Name: −container
Database Name: container
Database Class: Container

The value must be a boolean. If true, it means that this window will be used as a container in which
some other application will be embedded (for example, a Tk toplevel can be embedded using the −use
option). The window will support the appropriate window manager protocols for things like geometry
requests. The window should not have any children of its own in this application. This option may not be changed with the `configure` widget command.

**Command-Line Name:**  `-height`
**Database Name:**  `height`
**Database Class:**  `Height`

Specifies the desired height for the window in any of the forms acceptable to `Tk_GetPixels`. If this option is less than or equal to zero then the window will not request any size at all.

**Command-Line Name:**  `-menu`
**Database Name:**  `menu`
**Database Class:**  `Menu`

Specifies a menu widget to be used as a menubar. On the Macintosh, the menubar will be displayed across the top of the main monitor. On Microsoft Windows and all UNIX platforms, the menu will appear across the toplevel window as part of the window dressing maintained by the window manager.

**Command-Line Name:**  `-screen`
**Database Name:**
**Database Class:**

Specifies the screen on which to place the new window. Any valid screen name may be used, even one associated with a different display. Defaults to the same screen as its parent. This option is special in that it may not be specified via the option database, and it may not be modified with the `configure` widget command.

**Command-Line Name:**  `-use`
**Database Name:**  `use`
**Database Class:**  `Use`

This option is used for embedding. If the value isn't an empty string, it must be the window identifier of a container window, specified as a hexadecimal string like the ones returned by the `winfo id` command. The toplevel widget will be created as a child of the given container instead of the root window for the screen. If the container window is in a Tk application, it must be a frame or toplevel widget for which the `−container` option was specified. This option may not be changed with the `configure` widget command.

**Command-Line Name:**  `-visual`
**Database Name:**  `visual`
**Database Class:**  `Visual`

Specifies visual information for the new window in any of the forms accepted by `Tk_GetVisual`. If this option is not specified, the new window will use the default visual for its screen. The `visual` option may not be modified with the `configure` widget command.

**Command-Line Name:**  `−width`
**Database Name:**  `width`
**Database Class:**  `Width`

Specifies the desired width for the window in any of the forms acceptable to `Tk_GetPixels`. If this option is less than or equal to zero then the window will not request any size at all.
DESCRIPTION

The `toplevel` command creates a new toplevel widget (given by the `pathName` argument). Additional options, described above, may be specified on the command line or in the option database to configure aspects of the toplevel such as its background color and relief. The `toplevel` command returns the path name of the new window.

A toplevel is similar to a frame except that it is created as a top−level window: its X parent is the root window of a screen rather than the logical parent from its path name. The primary purpose of a toplevel is to serve as a container for dialog boxes and other collections of widgets. The only visible features of a toplevel are its background color and an optional 3−D border to make the toplevel appear raised or sunken.

WIDGET COMMAND

The `toplevel` command creates a new Tcl command whose name is the same as the path name of the toplevel's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the toplevel widget's path name. Option and the args determine the exact behavior of the command. The following commands are possible for toplevel widgets:

```
pathName cget option
```

Returns the current value of the configuration option given by option. Option may have any of the values accepted by the toplevel command.

```
pathName configure ?option? ?value option value ...?
```

Query or modify the configuration options of the widget. If no option is specified, returns a list describing all of the available options for pathName (see Tk_ConfigureInfo for information on the format of this list). If option is specified with no value, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no option is specified). If one or more option−value pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. Option may have any of the values accepted by the toplevel command.

BINDINGS

When a new toplevel is created, it has no default event bindings: toplevels are not intended to be interactive.

SEE ALSO

`frame`
KEYWORDS

toplevel, widget

Copyright ' 1990-1994 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

winfo − Return window−related information

SYNOPSIS

winfo option ?arg arg ...?

DESCRIPTION

winfo atom  ?−displayof window? name
winfo atomname  ?−displayof window? id
winfo cells window
winfo children window
winfo class window
winfo colormapfull window
winfo containing  ?−displayof window? rootX rootY
winfo depth window
winfo exists window
winfo fpixels window number
winfo geometry window
winfo height window
winfo id window
winfo interps  ?−displayof window?
winfo ismapped window
winfo manager window
winfo name window
winfo parent window
winfo pathname  ?−displayof window? id
winfo pixels window number
winfo pointerx window
winfo pointerxy window
winfo pointery window
winfo reqheight window
winfo reqwidth window
winfo rgb window color
winfo rootx window
winfo rooty window
winfo screen window
winfo screencells window
winfo screendumepth window
winfo screenheight window
winfo screenmmheight window
winfo screenmmwidth window
winfo screenvisual window
winfo screensewidth window
NAME

winfo – Return window-related information

SYNOPSIS

winfo option ?arg arg ...?

DESCRIPTION

The winfo command is used to retrieve information about windows managed by Tk. It can take any of a number of different forms, depending on the option argument. The legal forms are:

winfo atom ?–displayof window? name
Returns a decimal string giving the integer identifier for the atom whose name is name. If no atom exists with the name name then a new one is created. If the –displayof option is given then the atom is looked up on the display of window; otherwise it is looked up on the display of the application’s main window.

winfo atomname ?–displayof window? id
Returns the textual name for the atom whose integer identifier is id. If the –displayof option is given then the identifier is looked up on the display of window; otherwise it is looked up on the display of the application’s main window. This command is the inverse of the winfo atom command. It generates an error if no such atom exists.

winfo cells window
Returns a decimal string giving the number of cells in the color map for window.

winfo children window
Returns a list containing the path names of all the children of window. Top-level windows are returned as children of their logical parents. The list is in stacking order, with the lowest window first, except for Top-level windows which are not returned in stacking order. Use the wm stackorder command for further information.
command to query the stacking order of Top-level windows.

```tcl
winfo class window
```

Returns the class name for window.

```tcl
winfo colormapfull window
```

Returns 1 if the colormap for window is known to be full, 0 otherwise. The colormap for a window is "known" to be full if the last attempt to allocate a new color on that window failed and this application hasn't freed any colors in the colormap since the failed allocation.

```tcl
winfo containing ?-displayof window? rootX rootY
```

Returns the path name for the window containing the point given by rootX and rootY. RootX and rootY are specified in screen units (i.e. any form acceptable to `Tk_GetPixels`) in the coordinate system of the root window (if a virtual-root window manager is in use then the coordinate system of the virtual root window is used). If the `-displayof` option is given then the coordinates refer to the screen containing window; otherwise they refer to the screen of the application's main window. If no window in this application contains the point then an empty string is returned. In selecting the containing window, children are given higher priority than parents and among siblings the highest one in the stacking order is chosen.

```tcl
winfo depth window
```

Returns a decimal string giving the depth of window (number of bits per pixel).

```tcl
winfo exists window
```

Returns 1 if there exists a window named window, 0 if no such window exists.

```tcl
winfo fpixels window number
```

Returns a floating-point value giving the number of pixels in window corresponding to the distance given by number. Number may be specified in any of the forms acceptable to `Tk_GetScreenMM`, such as ``2.0c'' or ``1i''. The return value may be fractional; for an integer value, use `winfo pixels`.

```tcl
winfo geometry window
```

Returns the geometry for window, in the form widthxheight+x+y. All dimensions are in pixels.

```tcl
winfo height window
```

Returns a decimal string giving window's height in pixels. When a window is first created its height will be 1 pixel; the height will eventually be changed by a geometry manager to fulfill the window's needs. If you need the true height immediately after creating a widget, invoke `update` to force the geometry manager to arrange it, or use `winfo reqheight` to get the window's requested height instead of its actual height.

```tcl
winfo id window
```

Returns a hexadecimal string giving a low-level platform-specific identifier for window. On Unix platforms, this is the X window identifier. Under Windows, this is the Windows HWND. On the Macintosh the value has no meaning outside Tk.

```tcl
winfo interps ?,-displayof window?
```

Returns a list whose members are the names of all Tcl interpreters (e.g. all Tk-based applications) currently registered for a particular display. If the `-displayof` option is given then the return value refers to the display of window; otherwise it refers to the display of the application's main window.

```tcl
winfo ismapped window
```

Returns 1 if window is currently mapped, 0 otherwise.

```tcl
winfo manager window
```

Returns the name of the geometry manager currently responsible for window, or an empty string if window isn't managed by any geometry manager. The name is usually the name of the Tcl command
for the geometry manager, such as `pack` or `place`. If the geometry manager is a widget, such as canvases or text, the name is the widget's class command, such as `canvas`.

### `winfo name window`

Returns `window`'s name (i.e. its name within its parent, as opposed to its full path name). The command `winfo name .` will return the name of the application.

### `winfo parent window`

Returns the path name of `window`'s parent, or an empty string if `window` is the main window of the application.

### `winfo pathname ?–displayof window? id`

Returns the path name of the window whose X identifier is `id`. `Id` must be a decimal, hexadecimal, or octal integer and must correspond to a window in the invoking application. If the `-displayof` option is given then the identifier is looked up on the display of `window`; otherwise it is looked up on the display of the application's main window.

### `winfo pixels window number`

Returns the number of pixels in `window` corresponding to the distance given by `number`. `Number` may be specified in any of the forms acceptable to `Tk_GetPixels`, such as ``2.0c`` or ``1i``. The result is rounded to the nearest integer value; for a fractional result, use `winfo fpixels`.

### `winfo pointerx window`

If the mouse pointer is on the same screen as `window`, returns the pointer's x coordinate, measured in pixels in the screen's root window. If a virtual root window is in use on the screen, the position is measured in the virtual root. If the mouse pointer isn't on the same screen as `window` then -1 is returned.

### `winfo pointerxy window`

If the mouse pointer is on the same screen as `window`, returns a list with two elements, which are the pointer's x and y coordinates measured in pixels in the screen's root window. If a virtual root window is in use on the screen, the position is computed in the virtual root. If the mouse pointer isn't on the same screen as `window` then both of the returned coordinates are -1.

### `winfo pointery window`

If the mouse pointer is on the same screen as `window`, returns the pointer's y coordinate, measured in pixels in the screen's root window. If a virtual root window is in use on the screen, the position is computed in the virtual root. If the mouse pointer isn't on the same screen as `window` then -1 is returned.

### `winfo reqheight window`

Returns a decimal string giving `window`'s requested height, in pixels. This is the value used by `window`'s geometry manager to compute its geometry.

### `winfo reqwidth window`

Returns a decimal string giving `window`'s requested width, in pixels. This is the value used by `window`'s geometry manager to compute its geometry.

### `winfo rgb window color`

Returns a list containing three decimal values in the range 0 to 65535, which are the red, green, and blue intensities that correspond to `color` in the window given by `window`. `Color` may be specified in any of the forms acceptable for a color option.

### `winfo rootx window`

Returns a decimal string giving the x-coordinate, in the root window of the screen, of the upper-left corner of `window`'s border (or `window` if it has no border).
winfo rooty window
    Returns a decimal string giving the y−coordinate, in the root window of the screen, of the upper−left corner of window's border (or window if it has no border).

winfo screen window
    Returns the name of the screen associated with window, in the form displayName.screenIndex.

winfo screencells window
    Returns a decimal string giving the number of cells in the default color map for window's screen.

winfo screendepth window
    Returns a decimal string giving the depth of the root window of window's screen (number of bits per pixel).

winfo screenheight window
    Returns a decimal string giving the height of window's screen, in pixels.

winfo screenmmheight window
    Returns a decimal string giving the height of window's screen, in millimeters.

winfo screenmmwidth window
    Returns a decimal string giving the width of window's screen, in millimeters.

winfo screenvisual window
    Returns one of the following strings to indicate the default visual class for window's screen:
    directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor.

winfo screenwidth window
    Returns a decimal string giving the width of window's screen, in pixels.

winfo server window
    Returns a string containing information about the server for window's display. The exact format of this string may vary from platform to platform. For X servers the string has the form `"XmajorRminor vendor vendorVersion"` where major and minor are the version and revision numbers provided by the server (e.g., X11R5), vendor is the name of the vendor for the server, and vendorRelease is an integer release number provided by the server.

winfo toplevel window
    Returns the path name of the top−level window containing window.

winfo viewable window
    Returns 1 if window and all of its ancestors up through the nearest toplevel window are mapped. Returns 0 if any of these windows are not mapped.

winfo visual window
    Returns one of the following strings to indicate the visual class for window: directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor.

winfo visualid window
    Returns the X identifier for the visual for window.

winfo visualsavailable window ?includeids?
    Returns a list whose elements describe the visuals available for window's screen. Each element consists of a visual class followed by an integer depth. The class has the same form as returned by winfo visual. The depth gives the number of bits per pixel in the visual. In addition, if the includeids argument is provided, then the depth is followed by the X identifier for the visual.

winfo vrootheight window
    Returns the height of the virtual root window associated with window if there is one; otherwise returns the height of window's screen.
winfo vrootwidth window
Returns the width of the virtual root window associated with window if there is one; otherwise returns the width of window's screen.

winfo vrootx window
Returns the x−offset of the virtual root window associated with window, relative to the root window of its screen. This is normally either zero or negative. Returns 0 if there is no virtual root window for window.

winfo vrooty window
Returns the y−offset of the virtual root window associated with window, relative to the root window of its screen. This is normally either zero or negative. Returns 0 if there is no virtual root window for window.

winfo width window
Returns a decimal string giving window's width in pixels. When a window is first created its width will be 1 pixel; the width will eventually be changed by a geometry manager to fulfill the window's needs. If you need the true width immediately after creating a widget, invoke update to force the geometry manager to arrange it, or use winfo reqwidth to get the window's requested width instead of its actual width.

winfo x window
Returns a decimal string giving the x−coordinate, in window's parent, of the upper−left corner of window's border (or window if it has no border).

winfo y window
Returns a decimal string giving the y−coordinate, in window's parent, of the upper−left corner of window's border (or window if it has no border).

EXAMPLE

Print where the mouse pointer is and what window it is currently over:

set x [winfo pointerx .]
set y [winfo pointery .]
puts −nonewline "Mouse pointer at ($x,$y) which is "
set win [winfo containing $x $y]
if {$win eq ""} {
    puts "over no window"
} else {
    puts "over $win"
}

KEYWORDS

atom, children, class, geometry, height, identifier, information, interpreters, mapped, parent, path name, screen, virtual root, width, window

Copyright ' 1990−1994 The Regents of the University of California.
Copyright ' 1994−1997 Sun Microsystems, Inc.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
NAME

wm – Communicate with window manager

SYNOPSIS

wm option window ?args?

DESCRIPTION

wm aspect window ?minNumer minDenom maxNumer maxDenom?
wm attributes window
wm attributes window ?option?
wm attributes window ?option value option value...?
wm client window ?name?
wm colormapwindows window ?windowList?
wm command window ?value?
wm deiconify window
wm focusmodel window ?active|passive?
wm frame window
wm geometry window ?newGeometry?
wm grid window ?baseWidth baseHeight widthInc heightInc?
wm group window ?pathName?
wm iconbitmap window ?bitmap?
wm iconify window
wm iconmask window ?bitmap?
wm iconname window ?newName?
wm iconposition window ?x y?
wm iconwindow window ?pathName?
wm maxsize window ?width height?
wm minsize window ?width height?
wm overrideredirect window ?boolean?
wm positionfrom window ?who?
wm protocol window ?name? ?command?
wm resizable window ?width height?
wm sizefrom window ?who?
wm stackorder window ?isabove|isbelow window?
wm state window ?newstate?
wm title window ?string?
wm transient window ?master?
wm withdraw window

GEOMETRY MANAGEMENT
GRIDDED GEOMETRY MANAGEMENT

BUGS
EXAMPLES
SEE ALSO
**KEYWORDS**

**NAME**

wm – Communicate with window manager

**SYNOPSIS**

`wm option window ?args?`

**DESCRIPTION**

The `wm` command is used to interact with window managers in order to control such things as the title for a window, its geometry, or the increments in terms of which it may be resized. The `wm` command can take any of a number of different forms, depending on the `option` argument. All of the forms expect at least one additional argument, `window`, which must be the path name of a top–level window.

The legal forms for the `wm` command are:

`wm aspect window ?minNumer minDenom maxNumer maxDenom?`

If `minNumer`, `minDenom`, `maxNumer`, and `maxDenom` are all specified, then they will be passed to the window manager and the window manager should use them to enforce a range of acceptable aspect ratios for `window`. The aspect ratio of `window` (width/length) will be constrained to lie between `minNumer/minDenom` and `maxNumer/maxDenom`. If `minNumer` etc. are all specified as empty strings, then any existing aspect ratio restrictions are removed. If `minNumer` etc. are specified, then the command returns an empty string. Otherwise, it returns a Tcl list containing four elements, which are the current values of `minNumer`, `minDenom`, `maxNumer`, and `maxDenom` (if no aspect restrictions are in effect, then an empty string is returned).

`wm attributes window`

`wm attributes window ?option?`

`wm attributes window ?option value option value...?`

This subcommand returns or sets platform specific attributes associated with a window. The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:

On Windows, **–disabled** gets or sets whether the window is in a disabled state. **–toolwindow** gets or sets the style of the window to toolwindow (as defined in the MSDN). **–topmost** gets or sets whether this is a topmost window (displays above all other windows). **–alpha** sets the alpha transparency level of the toplevel. It accepts a value from 0.0 (fully transparent) to 1.0 (opaque). Values outside that range will be constrained. This is supported on Windows 2000/XP+. Where not supported, the **–alpha** value remains at 1.0.

On Mac OS X, **–modified** gets or sets the modification state of the window (determines whether the window close widget contains the modification indicator). **–titlepath** gets or sets
the path of the file referenced as the window proxy icon (which can be dragged and dropped in lieu of the file's finder icon). –alpha sets the alpha transparency level of the window, it accepts a value from 0.0 (fully transparent) to 1.0 (opaque), values outside that range will be constrained.

On Unix, there are currently no special attribute values.

wm client window ?name?
If name is specified, this command stores name (which should be the name of the host on which the application is executing) in window's WM_CLIENT_MACHINE property for use by the window manager or session manager. The command returns an empty string in this case. If name isn't specified, the command returns the last name set in a wm client command for window. If name is specified as an empty string, the command deletes the WM_CLIENT_MACHINE property from window.

wm colormapwindows window ?windowList?
This command is used to manipulate the WM_COLORMAP_WINDOWS property, which provides information to the window managers about windows that have private colormaps. If windowList isn't specified, the command returns a list whose elements are the names of the windows in the WM_COLORMAP_WINDOWS property. If windowList is specified, it consists of a list of window path names; the command overwrites the WM_COLORMAP_WINDOWS property with the given windows and returns an empty string. The WM_COLORMAP_WINDOWS property should normally contain a list of the internal windows within window whose colormaps differ from their parents. The order of the windows in the property indicates a priority order: the window manager will attempt to install as many colormaps as possible from the head of this list when window gets the colormap focus. If window is not included among the windows in windowList, Tk implicitly adds it at the end of the WM_COLORMAP_WINDOWS property, so that its colormap is lowest in priority. If wm colormapwindows is not invoked, Tk will automatically set the property for each top-level window to all the internal windows whose colormaps differ from their parents, followed by the top-level itself; the order of the internal windows is undefined. See the ICCCM documentation for more information on the WM_COLORMAP_WINDOWS property.

wm command window ?value?
If value is specified, this command stores value in window's WM_COMMAND property for use by the window manager or session manager and returns an empty string. Value must have proper list structure; the elements should contain the words of the command used to invoke the application. If value isn't specified then the command returns the last value set in a wm command command for window. If value is specified as an empty string, the command deletes the WM_COMMAND property from window.

wm deiconify window
Arrange for window to be displayed in normal (non-iconified) form. This is done by mapping the window. If the window has never been mapped then this command will not map the window, but it will ensure that when window is first mapped it will be displayed in de-iconified form. On Windows, a deiconified window will also be raised and be given the focus (made the active window). Returns an empty string.

wm focusmodel window ?active|passive?
If active or passive is supplied as an optional argument to the command, then it specifies the focus model for window. In this case the command returns an empty string. If no additional argument is
supplied, then the command returns the current focus model for window. An active focus model means that window will claim the input focus for itself or its descendants, even at times when the focus is currently in some other application. Passive means that window will never claim the focus for itself; the window manager should give the focus to window at appropriate times. However, once the focus has been given to window or one of its descendants, the application may re-assign the focus among window's descendants. The focus model defaults to passive, and Tk's focus command assumes a passive model of focusing.

wm frame window
If window has been reparented by the window manager into a decorative frame, the command returns the platform specific window identifier for the outermost frame that contains window (the window whose parent is the root or virtual root). If window hasn't been reparented by the window manager then the command returns the platform specific window identifier for window.

wm geometry window ?newGeometry?
If newGeometry is specified, then the geometry of window is changed and an empty string is returned. Otherwise the current geometry for window is returned (this is the most recent geometry specified either by manual resizing or in a wm geometry command). NewGeometry has the form =widthxheight–x–y, where any of =, widthxheight, or –x–y may be omitted. Width and height are positive integers specifying the desired dimensions of window. If window is gridded (see GRIDDED GEOMETRY MANAGEMENT below) then the dimensions are specified in grid units; otherwise they are specified in pixel units. X and y specify the desired location of window on the screen, in pixels. If x is preceded by +, it specifies the number of pixels between the left edge of the screen and the left edge of window's border; if preceded by – then x specifies the number of pixels between the right edge of the screen and the right edge of window's border. If y is preceded by + then it specifies the number of pixels between the top of the screen and the top of window's border; if y is preceded by – then it specifies the number of pixels between the bottom of window's border and the bottom of the screen. If newGeometry is specified as an empty string then any existing user-specified geometry for window is cancelled, and the window will revert to the size requested internally by its widgets.

wm grid window ?baseWidth baseHeight widthInc heightInc?
This command indicates that window is to be managed as a gridded window. It also specifies the relationship between grid units and pixel units. BaseWidth and baseHeight specify the number of grid units corresponding to the pixel dimensions requested internally by window using Tk_GeometryRequest. WidthInc and heightInc specify the number of pixels in each horizontal and vertical grid unit. These four values determine a range of acceptable sizes for window, corresponding to grid-based widths and heights that are non-negative integers. Tk will pass this information to the window manager; during manual resizing, the window manager will restrict the window's size to one of these acceptable sizes. Furthermore, during manual resizing the window manager will display the window's current size in terms of grid units rather than pixels. If baseWidth etc. are all specified as empty strings, then window will no longer be managed as a gridded window. If baseWidth etc. are specified then the return value is an empty string. Otherwise the return value is a Tcl list containing four elements corresponding to the current baseWidth, baseHeight, widthInc, and heightInc; if window is not currently gridded, then an empty string is returned. Note; this command should not be needed very often, since the Tk_SetGrid library procedure and the setGrid option provide easier access to the same functionality.

wm group window ?pathName?
If pathName is specified, it gives the path name for the leader of a group of related windows. The
window manager may use this information, for example, to unmap all of the windows in a group when the group's leader is iconified. PathName may be specified as an empty string to remove window from any group association. If pathName is specified then the command returns an empty string; otherwise it returns the path name of window's current group leader, or an empty string if window isn't part of any group.

wm iconbitmap window ?bitmap?
If bitmap is specified, then it names a bitmap in the standard forms accepted by Tk (see the Tk_GetBitmap manual entry for details). This bitmap is passed to the window manager to be displayed in window's icon, and the command returns an empty string. If an empty string is specified for bitmap, then any current icon bitmap is cancelled for window. If bitmap is specified then the command returns an empty string. Otherwise it returns the name of the current icon bitmap associated with window, or an empty string if window has no icon bitmap. On the Windows operating system, an additional flag is supported: wm iconbitmap window ?−default? ?image?. If the −default flag is given, the icon is applied to all toplevel windows (existing and future) to which no other specific icon has yet been applied. In addition to bitmap image types, a full path specification to any file which contains a valid Windows icon is also accepted (usually .ico or .icr files), or any file for which the shell has assigned an icon. Tcl will first test if the file contains an icon, then if it has an assigned icon, and finally, if that fails, test for a bitmap.

wm iconify window
Arragne for window to be iconified. If window hasn't yet been mapped for the first time, this command will arrange for it to appear in the iconified state when it is eventually mapped.

wm iconmask window ?bitmap?
If bitmap is specified, then it names a bitmap in the standard forms accepted by Tk (see the Tk_GetBitmap manual entry for details). This bitmap is passed to the window manager to be used as a mask in conjunction with the iconbitmap option: where the mask has zeroes no icon will be displayed; where it has ones, the bits from the icon bitmap will be displayed. If an empty string is specified for bitmap then any current icon mask is cancelled for window (this is equivalent to specifying a bitmap of all ones). If bitmap is specified then the command returns an empty string. Otherwise it returns the name of the current icon mask associated with window, or an empty string if no mask is in effect.

wm iconname window ?newName?
If newName is specified, then it is passed to the window manager; the window manager should display newName inside the icon associated with window. In this case an empty string is returned as result. If newName isn't specified then the command returns the current icon name for window, or an empty string if no icon name has been specified (in this case the window manager will normally display the window's title, as specified with the wm title command).

wm iconposition window ?x y?
If x and y are specified, they are passed to the window manager as a hint about where to position the icon for window. In this case an empty string is returned. If x and y are specified as empty strings then any existing icon position hint is cancelled. If neither x nor y is specified, then the command returns a Tcl list containing two values, which are the current icon position hints (if no hints are in effect then an empty string is returned).

wm iconwindow window ?pathName?
If pathName is specified, it is the path name for a window to use as icon for window: when window is iconified then pathName will be mapped to serve as icon, and when window is de–iconified then
pathName will be unmapped again. If pathName is specified as an empty string then any existing icon window association for window will be cancelled. If the pathName argument is specified then an empty string is returned. Otherwise the command returns the path name of the current icon window for window, or an empty string if there is no icon window currently specified for window. Button press events are disabled for window as long as it is an icon window; this is needed in order to allow window managers to ``own'' those events. Note: not all window managers support the notion of an icon window.

wm maxsize window ?width height?
If width and height are specified, they give the maximum permissible dimensions for window. For gridded windows the dimensions are specified in grid units; otherwise they are specified in pixel units. The window manager will restrict the window's dimensions to be less than or equal to width and height. If width and height are specified, then the command returns an empty string. Otherwise it returns a Tcl list with two elements, which are the maximum width and height currently in effect. The maximum size defaults to the size of the screen. See the sections on geometry management below for more information.

wm minsize window ?width height?
If width and height are specified, they give the minimum permissible dimensions for window. For gridded windows the dimensions are specified in grid units; otherwise they are specified in pixel units. The window manager will restrict the window's dimensions to be greater than or equal to width and height. If width and height are specified, then the command returns an empty string. Otherwise it returns a Tcl list with two elements, which are the minimum width and height currently in effect. The minimum size defaults to one pixel in each dimension. See the sections on geometry management below for more information.

wm overrideredirect window ?boolean?
If boolean is specified, it must have a proper boolean form and the override−redirect flag for window is set to that value. If boolean is not specified then 1 or 0 is returned to indicate whether or not the override−redirect flag is currently set for window. Setting the override−redirect flag for a window causes it to be ignored by the window manager; among other things, this means that the window will not be reparented from the root window into a decorative frame and the user will not be able to manipulate the window using the normal window manager mechanisms.

wm positionfrom window ?who?
If who is specified, it must be either program or user, or an abbreviation of one of these two. It indicates whether window's current position was requested by the program or by the user. Many window managers ignore program−requested initial positions and ask the user to manually position the window; if user is specified then the window manager should position the window at the given place without asking the user for assistance. If who is specified as an empty string, then the current position source is cancelled. If who is specified, then the command returns an empty string. Otherwise it returns user or program to indicate the source of the window's current position, or an empty string if no source has been specified yet. Most window managers interpret ”no source” as equivalent to program. Tk will automatically set the position source to user when a wm geometry command is invoked, unless the source has been set explicitly to program.

wm protocol window ?name? ?command?
This command is used to manage window manager protocols such as WM_DELETE_WINDOW. Name is the name of an atom corresponding to a window manager protocol, such as WM_DELETE_WINDOW or WM_SAVE_YOURSELF or WM_TAKE_FOCUS. If both name
and \textit{command} are specified, then \textit{command} is associated with the protocol specified by \textit{name}. \textit{Name} will be added to \textit{window}'s \texttt{WM_PROTOCOLS} property to tell the window manager that the application has a protocol handler for \textit{name}, and \textit{command} will be invoked in the future whenever the window manager sends a message to the client for that protocol. In this case the command returns an empty string. If \textit{name} is specified but \textit{command} isn't, then the current command for \textit{name} is returned, or an empty string if there is no handler defined for \textit{name}. If \textit{command} is specified as an empty string then the current handler for \textit{name} is deleted and it is removed from the \texttt{WM_PROTOCOLS} property on \textit{window}; an empty string is returned. Lastly, if neither \textit{name} nor \textit{command} is specified, the command returns a list of all the protocols for which handlers are currently defined for \textit{window}.

Tk always defines a protocol handler for \texttt{WM_DELETE_WINDOW}, even if you haven't asked for one with \texttt{wm protocol}. If a \texttt{WM_DELETE_WINDOW} message arrives when you haven't defined a handler, then Tk handles the message by destroying the window for which it was received.

\texttt{wm resizable} \textit{window} ?\textit{width} \textit{height}?

This command controls whether or not the user may interactively resize a top–level window. If \textit{width} and \textit{height} are specified, they are boolean values that determine whether the width and height of \textit{window} may be modified by the user. In this case the command returns an empty string. If \textit{width} and \textit{height} are omitted then the command returns a list with two 0/1 elements that indicate whether the width and height of \textit{window} are currently resizable. By default, windows are resizable in both dimensions. If resizing is disabled, then the window's size will be the size from the most recent interactive resize or \texttt{wm geometry} command. If there has been no such operation then the window's natural size will be used.

\texttt{wm sizefrom} \textit{window} ?\textit{who}?

If \textit{who} is specified, it must be either \texttt{program} or \texttt{user}, or an abbreviation of one of these two. It indicates whether \textit{window}'s current size was requested by the program or by the user. Some window managers ignore program–requested sizes and ask the user to manually size the window; if \texttt{user} is specified then the window manager should give the window its specified size without asking the user for assistance. If \textit{who} is specified as an empty string, then the current size source is cancelled. If \textit{who} is specified, then the command returns an empty string. Otherwise it returns \texttt{user} or \texttt{window} to indicate the source of the window's current size, or an empty string if no source has been specified yet. Most window managers interpret "no source" as equivalent to \texttt{program}.

\texttt{wm stackorder} \textit{window} ?\texttt{isabove} \texttt{isbelow} \textit{window}?

The stackorder command returns a list of toplevel windows in stacking order, from lowest to highest. When a single toplevel window is passed, the returned list recursively includes all of the window's children that are toplevels. Only those toplevels that are currently mapped to the screen are returned. The stackorder command can also be used to determine if one toplevel is positioned above or below a second toplevel. When two window arguments separated by either \texttt{isabove} or \texttt{isbelow} are passed, a boolean result indicates whether or not the first window is currently above or below the second window in the stacking order.

\texttt{wm state} \textit{window} ?\texttt{newstate}?

If \textit{newstate} is specified, the window will be set to the new state, otherwise it returns the current state of \textit{window}: either \texttt{normal}, \texttt{iconic}, \texttt{withdrawn}, \texttt{icon}, or (Windows only) \texttt{zoomed}. The difference between \texttt{iconic} and \texttt{icon} is that \texttt{iconic} refers to a window that has been iconified (e.g., with the \texttt{wm iconify} command) while \texttt{icon} refers to a window whose only purpose is to serve as the icon for some other window (via the \texttt{wm iconwindow} command). The \texttt{icon} state cannot be set.
WM TITLE window ?string?

If string is specified, then it will be passed to the window manager for use as the title for window (the window manager should display this string in window's title bar). In this case the command returns an empty string. If string isn't specified then the command returns the current title for the window. The title for a window defaults to its name.

WM TRANSIENT window ?master?

If master is specified, then the window manager is informed that window is a transient window (e.g. pull−down menu) working on behalf of master (where master is the path name for a top−level window). If master is specified as an empty string then window is marked as not being a transient window any more. Otherwise the command returns the path name of window's current master, or an empty string if window isn't currently a transient window. A transient window will mirror state changes in the master and inherit the state of the master when initially mapped. It is an error to attempt to make a window a transient of itself.

WM WITHDRAW window

Arranges for window to be withdrawn from the screen. This causes the window to be unmapped and forgotten about by the window manager. If the window has never been mapped, then this command causes the window to be mapped in the withdrawn state. Not all window managers appear to know how to handle windows that are mapped in the withdrawn state. Note: it sometimes seems to be necessary to withdraw a window and then re−map it (e.g. with wm deiconify) to get some window managers to pay attention to changes in window attributes such as group.

GEOMETRY MANAGEMENT

By default a top−level window appears on the screen in its natural size, which is the one determined internally by its widgets and geometry managers. If the natural size of a top−level window changes, then the window's size changes to match. A top−level window can be given a size other than its natural size in two ways. First, the user can resize the window manually using the facilities of the window manager, such as resize handles. Second, the application can request a particular size for a top−level window using the wm geometry command. These two cases are handled identically by Tk; in either case, the requested size overrides the natural size. You can return the window to its natural by invoking wm geometry with an empty geometry string.

Normally a top−level window can have any size from one pixel in each dimension up to the size of its screen. However, you can use the wm minsize and wm maxsize commands to limit the range of allowable sizes. The range set by wm minsize and wm maxsize applies to all forms of resizing, including the window's natural size as well as manual resizes and the wm geometry command. You can also use the command wm resizable to completely disable interactive resizing in one or both dimensions.

GRIDDED GEOMETRY MANAGEMENT

Gridded geometry management occurs when one of the widgets of an application supports a range of useful sizes. This occurs, for example, in a text editor where the scrollbars, menus, and other adornments are fixed in size but the edit widget can support any number of lines of text or characters per line. In this case, it is usually desirable to let the user specify the number of lines or characters−per−line, either with the wm geometry command or by interactively resizing the window. In the case of text, and in other interesting cases also, only
discrete sizes of the window make sense, such as integral numbers of lines and characters–per–line; arbitrary pixel sizes are not useful.

Gridded geometry management provides support for this kind of application. Tk (and the window manager) assume that there is a grid of some sort within the application and that the application should be resized in terms of grid units rather than pixels. Gridded geometry management is typically invoked by turning on the setGrid option for a widget; it can also be invoked with the wm grid command or by calling Tk_SetGrid. In each of these approaches the particular widget (or sometimes code in the application as a whole) specifies the relationship between integral grid sizes for the window and pixel sizes. To return to non–gridded geometry management, invoke wm grid with empty argument strings.

When gridded geometry management is enabled then all the dimensions specified in wm minsize, wm maxsize, and wm geometry commands are treated as grid units rather than pixel units. Interactive resizing is also carried out in even numbers of grid units rather than pixels.

**BUGS**

Most existing window managers appear to have bugs that affect the operation of the wm command. For example, some changes won’t take effect if the window is already active: the window will have to be withdrawn and de–iconified in order to make the change happen.

**EXAMPLES**

A fixed–size window that says that it is fixed–size too:

toplevel .fixed
wm title .fixed "Fixed–size Window"
wm resizable .fixed 0 0

A simple dialog–like window, centred on the screen:

```
# Create and arrange the dialog contents.
toplevel .msg
label .msg.l -text "This is a very simple dialog demo."
button .msg.ok -text OK -default active -command {destroy .msg}
pack .msg.ok -side bottom -fill x
pack .msg.l -expand 1 -fill both

# Now set the widget up as a centred dialog.

# But first, we need the geometry managers to finish setting
# up the interior of the dialog, for which we need to run the
# event loop with the widget hidden completely...
wm withdraw .msg
update
set x [expr {([winfo screenwidth .]−[winfo width .msg])/2}]
set y [expr {([winfo screenheight .]−[winfo height .msg])/2}]
wm geometry .msg +$x+$y
wm transient .msg
```
wm title .msg "Dialog demo"
wmd deiconify .msg

SEE ALSO

toplevel, winfo

KEYWORDS

aspect ratio, deiconify, focus model, geometry, grid, group, icon, iconify, increments, position, size, title, top−level window, units, window manager

Copyright ' 1991−1994 The Regents of the University of California.
Copyright ' 1994−1996 Sun Microsystems, Inc.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcl_detachChannel</td>
<td>Detach a channel</td>
</tr>
<tr>
<td>Tcl_detachPids</td>
<td>Detach pids</td>
</tr>
<tr>
<td>Tcl_discardResult</td>
<td>Discard a result</td>
</tr>
<tr>
<td>Tcl_doOneEvent</td>
<td>Invoke an event</td>
</tr>
<tr>
<td>Tcl_doWhenIdle</td>
<td>Do when idle</td>
</tr>
<tr>
<td>Tcl_dontCallWhenDeleted</td>
<td>Don't call when deleted</td>
</tr>
<tr>
<td>Tcl_dumpActiveMemory</td>
<td>Dump active memory</td>
</tr>
<tr>
<td>Tcl_duplicateObj</td>
<td>Duplicate an object</td>
</tr>
<tr>
<td>Tcl_eof</td>
<td>End of file</td>
</tr>
<tr>
<td>Tcl_errnoId</td>
<td>Error number</td>
</tr>
<tr>
<td>Tcl_errnoMsg</td>
<td>Error message</td>
</tr>
<tr>
<td>Tcl_eval</td>
<td>Evaluate an expression</td>
</tr>
<tr>
<td>Tcl_evalEx</td>
<td>Evaluate an expression with extensions</td>
</tr>
<tr>
<td>Tcl_evalFile</td>
<td>Evaluate an expression file</td>
</tr>
<tr>
<td>Tcl_evalObjEx</td>
<td>Evaluate an expression object</td>
</tr>
<tr>
<td>Tcl_evalObjv</td>
<td>Evaluate an expression vector</td>
</tr>
<tr>
<td>Tcl_evalTokens</td>
<td>Evaluate an expression tokens</td>
</tr>
<tr>
<td>Tcl_getNameOfExecutable</td>
<td>Get name of executable</td>
</tr>
<tr>
<td>Tcl_getOSTypeFromObj</td>
<td>Get operating system type</td>
</tr>
<tr>
<td>Tcl_getObjResult</td>
<td>Get object result</td>
</tr>
<tr>
<td>Tcl_getObjType</td>
<td>Get object type</td>
</tr>
<tr>
<td>Tcl_getOpenFile</td>
<td>Get open file</td>
</tr>
<tr>
<td>Tcl_getPathType</td>
<td>Get path type</td>
</tr>
<tr>
<td>Tcl_getRange</td>
<td>Get range</td>
</tr>
<tr>
<td>Tcl_getRegExpFromObj</td>
<td>Get regular expression from object</td>
</tr>
<tr>
<td>Tcl_getRegExpRange</td>
<td>Get regular expression range</td>
</tr>
<tr>
<td>Tcl_getSlave</td>
<td>Get slave</td>
</tr>
<tr>
<td>Tcl_getStackedChannel</td>
<td>Get stacked channel</td>
</tr>
<tr>
<td>Tcl_getStdChannel</td>
<td>Get standard channel</td>
</tr>
<tr>
<td>Tcl_getString</td>
<td>Get string</td>
</tr>
<tr>
<td>Tcl_getStringFromObj</td>
<td>Get string from object</td>
</tr>
<tr>
<td>Tcl_getStringResult</td>
<td>Get string result</td>
</tr>
<tr>
<td>Tcl_getThreadData</td>
<td>Get thread data</td>
</tr>
<tr>
<td>Tcl_getTime</td>
<td>Get time</td>
</tr>
<tr>
<td>Tcl_registerChannel</td>
<td>Register a channel</td>
</tr>
<tr>
<td>Tcl_registerObjType</td>
<td>Register an object type</td>
</tr>
<tr>
<td>Tcl_release</td>
<td>Release an object</td>
</tr>
<tr>
<td>Tcl_resetResult</td>
<td>Reset result</td>
</tr>
<tr>
<td>Tcl_restoreResult</td>
<td>Restore result</td>
</tr>
<tr>
<td>Tcl_saveResult</td>
<td>Save result</td>
</tr>
<tr>
<td>Tcl_scanCountedElement</td>
<td>Scan counted element</td>
</tr>
<tr>
<td>Tcl_scanElement</td>
<td>Scan element</td>
</tr>
<tr>
<td>Tcl_seek</td>
<td>Seek</td>
</tr>
<tr>
<td>Tcl_serviceAll</td>
<td>Service all</td>
</tr>
<tr>
<td>Tcl_serviceEvent</td>
<td>Service event</td>
</tr>
<tr>
<td>Tcl_setAssocData</td>
<td>Set assoc data</td>
</tr>
<tr>
<td>Tcl_setBooleanObj</td>
<td>Set boolean object</td>
</tr>
<tr>
<td>Tcl_setByteArrayLength</td>
<td>Set byte array length</td>
</tr>
<tr>
<td>Tcl_setByteArrayObj</td>
<td>Set byte array object</td>
</tr>
<tr>
<td>Tcl_winUtfToTChar</td>
<td>Convert UTF-8 to TCHAR</td>
</tr>
<tr>
<td>Tcl_write</td>
<td>Write</td>
</tr>
<tr>
<td>Tcl_writeChars</td>
<td>Write characters</td>
</tr>
<tr>
<td>Tcl_writeRaw</td>
<td>Write raw</td>
</tr>
<tr>
<td>Tcl_wrongNumArgs</td>
<td>Wrong number of arguments</td>
</tr>
</tbody>
</table>

Copyright © 1989-1994 The Regents of the University of California.
Copyright © 1992-1999 Karl Lehenbauer and Mark Diekhans.
Copyright © 1994-1998 Sun Microsystems, Inc.
Copyright © 1997-2000 Ajuba Solutions.
Copyright © 1998-1999 Scriptics Corporation
Copyright © 2001 ActiveState Corporation
Copyright © 2001 ActiveState Tool Corp.
Copyright © 2001 Kevin B. Kenny.
Copyright © 2001 Vincent Darley
Copyright © 2001-2002 Donal K. Fellows
Copyright © 2002 Kevin B. Kenny. All rights reserved.
Copyright © 1995-1997 Roger E. Critchlow Jr.
NAME

Tcl_FSRegister, Tcl_FSUnregister, Tcl_FSData, Tcl_FSMountsChanged,
Tcl_FSGetFileSystemForPath, Tcl_FSGetPathType, Tcl_FSCopyFile, Tcl_FSCopyDirectory,
Tcl_FSCreateDirectory, Tcl_FSDeleteFile, Tcl_FSRemoveDirectory, Tcl_FSRenameFile,
Tcl_FSLListVolumes, Tcl_FSResolveFile, Tcl_FSLoadFile, Tcl_FSMatchInDirectory,
Tcl_FSLink, Tcl_FSLstat, Tcl_FSUtime, Tcl_FSFileAttrsGet, Tcl_FSFileAttrsSet,
Tcl_FSFileAttrStrings, Tcl_FSSstat, Tcl_FSAccess, Tcl_FSOpendFileChannel,
Tcl_FSGetCwd, Tcl_FSChdir, Tcl_FSPathSeparator, Tcl_FSJoinPath, Tcl_FSPathSplitPath,
Tcl_FSEqualPaths, Tcl_FSGetNormalizedPath, Tcl_FSJoinToPath,
Tcl_FSConvertToPathType, Tcl_FSGetInternalRep, Tcl_FSGetTranslatedPath,
Tcl_FSGetTranslatedStringPath, Tcl_FSNewNativePath, Tcl_FSGetNativePath,
Tcl_FSFileSystemInfo, Tcl_AllocStatBuf – procedures to interact with any filesystem

SYNOPSIS

ARGUMENTS

DESCRIPTION

TCL_FILESYSTEM

TYPENAME

STRUCTURE LENGTH

VERSION

FILESYSTEM INFRASTRUCTURE

PATHINFFILESYSTEMPROC

DUPINTERNALREPPROC

FREEINTERNALREPPROC

INTERNALTONORMALIZEDPROC

CREATEINTERNALREPPROC

NORMALIZEPATHPROC

FILESYSTEM OPERATIONS

FILESYSTEMPATHTYPEPROC

FILESYSTEMSEPARATORPROC

STATPROC

ACCESSPROC

OPENFILECHANNELPROC

MATCHINDIRECTORYPROC

UTIMEPROC

LINKPROC

LISTVOLUMESPROC

FILEATTRSTRINGSPROC

FILEATTRSGETPROC

FILEATTRSSETPROC

CREATEDIRECTORYPROC

REMOVEDIRECTORYPROC
DELETEFILEPROC
FILESYSTEM EFFICIENCY
LSTATPROC
COPYFILEPROC
RENAMEFILEPROC
COPYDIRECTORYPROC
LOADFILEPROC
UNLOADFILEPROC
GETCWDPROC
CHDIRPROC
KEYWORDS

NAME

Tcl_FSRegister, Tcl_FSUnregister, Tcl_FSData, Tcl_FSMountsChanged, Tcl_FSGetFileSystemForPath, Tcl_FSGetPathType, Tcl_FSCopyFile, Tcl_FSCopyDirectory, Tcl_FSCreateDirectory, Tcl_FSDeleteFile, Tcl_FSRenameFile, Tcl_FSLlistVolumes, Tcl_FSEvalFile, Tcl_FSLoadFile, Tcl_FSMatchInDirectory, Tcl_FSLink, Tcl_FSLstat, Tcl_FSUtime, Tcl_FSFileAttrsGet, Tcl_FSFileAttrsSet, Tcl_FSFileAttrStrings, Tcl_FSStat, Tcl_FSAccess, Tcl_FSOpenFileChannel, Tcl_FSGetCwd, Tcl_FSChdir, Tcl_FSPathSeparator, Tcl_FSSJoinPath, Tcl_FSJoinPath, Tcl_FSSplitPath, Tcl_FSEqualPaths, Tcl_FSGetNormalizedPath, Tcl_FSVJoinPath, Tcl_FSVConToPathType, Tcl_FSGetsInternalRep, Tcl_FSGetTranslatedPath, Tcl_FSGetTranslatedStringPath, Tcl_FSNewNativePath, Tcl_FSGetNativePath, Tcl_FSFileSystemInfo, Tcl_AllocStatBuf – procedures to interact with any filesystem

SYNOPSIS

#include <tcl.h>
int
Tcl_FSRegister(clientData, fsPtr)
int
Tcl_FSUnregister(fsPtr)
ClientData
Tcl_FSData(fsPtr)
void
Tcl_FSMountsChanged(fsPtr)
Tcl_Filesystem*
Tcl_FSGetFileSystemForPath(pathObjPtr)
Tcl_PathType
Tcl_FSGetPathType(pathObjPtr)
int
Tcl_FSCopyFile(srcPathPtr, destPathPtr)
int
Tcl_FSCopyDirectory(srcPathPtr, destPathPtr, errorPtr)
int
Tcl_FSCreateDirectory(pathPtr)
int Tcl_FSDeleteFile(pathPtr)
int Tcl_FSRemoveDirectory(pathPtr, int recursive, errorPtr)
int Tcl_FSRenameFile(srcPathPtr, destPathPtr)
Tcl_Obj* Tcl_FSListVolumes(void)
int Tcl_FSEvalFile(interp, pathPtr)
int Tcl_FSLoadFile(interp, pathPtr, sym1, sym2, proc1Ptr, proc2Ptr, handlePtr, unloadProcPtr)
int Tcl_FSMatchInDirectory(interp, result, pathPtr, pattern, types)
Tcl_Obj* Tcl_FSLink(linkNamePtr, toPtr, linkAction)
int Tcl_FSLstat(pathPtr, statPtr)
int Tcl_FSUtime(pathPtr, tval)
int Tcl_FSFileAttrsGet(interp, int index, pathPtr, objPtrRef)
int Tcl_FSFileAttrsSet(interp, int index, pathPtr, Tcl_Obj *objPtr)
CONST char** Tcl_FSFileAttrStrings(pathPtr, objPtrRef)
int Tcl_FSSStat(pathPtr, statPtr)
int Tcl_FSAccess(pathPtr, mode)
Tcl_Channel Tcl_FSOOpenFileChannel(interp, pathPtr, modeString, permissions)
Tcl_Obj* Tcl_FSGetCwd(interp)
int Tcl_FSChdir(pathPtr)
Tcl_Obj* Tcl_FSPATHSeparator(pathPtr)
Tcl_Obj* Tcl_FSSJoinPath(listObj, elements)
Tcl_Obj* Tcl_FSSSplitPath(pathPtr, lenPtr)
int Tcl_FSEqualPaths(firstPtr, secondPtr)
Tcl_Obj*
ARGUMENTS

Tcl_Filesystem *fsPtr (in)
Points to a structure containing the addresses of procedures that can be called to perform the various
filesystem operations.

Tcl_Obj *pathPtr (in)
The path represented by this object is used for the operation in question. If the object does not already
have an internal path representation, it will be converted to have one.

Tcl_Obj *srcPathPtr (in)
As for pathPtr, but used for the source file for a copy or rename operation.

Tcl_Obj *destPathPtr (in)
As for pathPtr, but used for the destination filename for a copy or rename operation.

CONST char *pattern (in)
Only files or directories matching this pattern will be returned by Tcl_FSMatchInDirectory.

GlobTypeData *types (in)
Only files or directories matching the type descriptions contained in this structure will be returned by
Tcl_FSMatchInDirectory. It is very important that the 'directory' flag is properly handled. This
parameter may be NULL.

Tcl_Interp *interp (in)
Interpreter to use either for results, evaluation, or reporting error messages.

ClientData clientData (in)
The native description of the path object to create.

Tcl_Obj *firstPtr (in)
The first of two path objects to compare. The object may be converted to path type.

Tcl_Obj *secondPtr (in)
The second of two path objects to compare. The object may be converted to path type.

*Tcl_Obj *listObj (in)*

The list of path elements to operate on with a join operation.

*int elements (in)*

If non-negative, the number of elements in the listObj which should be joined together. If negative, then all elements are joined.

*Tcl_Obj **errorPtr (out)*

In the case of an error, filled with an object containing the name of the file which caused an error in the various copy/rename operations.

*Tcl_Obj **objPtrRef (out)*

Filled with an object containing the result of the operation.

*Tcl_Obj *result (out)*

Pre-allocated object in which to store (by lappending) the list of files or directories which are successfully matched in Tcl_FSMatchInDirectory.

*int mode (in)*

Mask consisting of one or more of R_OK, W_OK, X_OK and F_OK. R_OK, W_OK and X_OK request checking whether the file exists and has read, write and execute permissions, respectively. F_OK just requests checking for the existence of the file.

*Tcl_StatBuf *statPtr (out)*

The structure that contains the result of a stat or lstat operation.

*CONST char *sym1 (in)*

Name of a procedure to look up in the file's symbol table

*CONST char *sym2 (in)*

Name of a procedure to look up in the file's symbol table

*Tcl_PackageInitProc **proc1Ptr (out)*

Filled with the init function for this code.

*Tcl_PackageInitProc **proc2Ptr (out)*

Filled with the safe-init function for this code.

*ClientData *clientDataPtr (out)*

Filled with the clientData value to pass to this code's unload function when it is called.

*TclfsUnloadFileProc_ **unloadProcPtr (out)*

Filled with the function to use to unload this piece of code.

*utimbuf *tval (in)*

The access and modification times in this structure are read and used to set those values for a given file.

*CONST char *modeString (in)*

Specifies how the file is to be accessed. May have any of the values allowed for the mode argument to the Tcl open command.

*int permissions (in)*

POSIX-style permission flags such as 0644. If a new file is created, these permissions will be set on the created file.

*int *lenPtr (out)*

If non-NULL, filled with the number of elements in the split path.

*Tcl_Obj *basePtr (in)*

The base path on to which to join the given elements. May be NULL.
The number of elements in \textit{objv}.

Tcl\_Obj *\textit{objv}[] (in)

The elements to join to the given base path.

\section*{DESCRIPTION}

There are several reasons for calling the \texttt{Tcl\_FS\...} functions rather than calling system level functions like \texttt{access} and \texttt{stat} directly. First, they will work cross-platform, so an extension which calls them should work unmodified on Unix, MacOS and Windows. Second, the Windows implementation of some of these functions fixes some bugs in the system level calls. Third, these function calls deal with any 'Utf to platform–native' path conversions which may be required (and may cache the results of such conversions for greater efficiency on subsequent calls). Fourth, and perhaps most importantly, all of these functions are 'virtual filesystem aware'. Any virtual filesystem which has been registered (through \texttt{Tcl\_FSRegister}) may reroute file access to alternative media or access methods. This means that all of these functions (and therefore the corresponding \texttt{file}, \texttt{glob}, \texttt{pwd}, \texttt{cd}, \texttt{open}, etc. Tcl commands) may be operate on 'files' which are not native files in the native filesystem. This also means that any Tcl extension which accesses the filesystem through this API is automatically 'virtual filesystem aware'. Of course, if an extension accesses the native filesystem directly (through platform−specific APIs, for example), then Tcl cannot intercept such calls.

If appropriate vfs's have been registered, the 'files' may, to give two examples, be remote (e.g. situated on a remote ftp server) or archived (e.g. lying inside a .zip archive). Such registered filesystems provide a lookup table of functions to implement all or some of the functionality listed here. Finally, the \texttt{Tcl\_FSSStat} and \texttt{Tcl\_FSLstat} calls abstract away from what the 'struct stat' buffer buffer is actually declared to be, allowing the same code to be used both on systems with and systems without support for files larger than 2GB in size.

The \texttt{Tcl\_FS\...} are objectified and may cache internal representations and other path−related strings (e.g. the current working directory). One side−effect of this is that one must not pass in objects with a refCount of zero to any of these functions. If such calls were handled, they might result in memory leaks (under some circumstances, the filesystem code may wish to retain a reference to the passed in object, and so one must not assume that after any of these calls return, the object still has a refCount of zero – it may have been incremented), or in a direct segfault due to the object being freed part way through the complex object manipulation required to ensure that the path is fully normalized and absolute for filesystem determination. The practical lesson to learn from this is that \texttt{Tcl\_Obj *}\texttt{path} = \texttt{Tcl\_NewStringObj}(...) ; \texttt{Tcl\_FS\...}(\texttt{path}) ; \texttt{Tcl\_DecrRefCount(path)} is wrong, and may segfault. The 'path' must have its refCount incremented before passing it in, or decrementing it. For this reason, objects with a refCount of zero are considered not to be valid filesystem paths and calling any Tcl\_FS API with such an object will result in no action being taken.

\texttt{Tcl\_FSCopyFile} attempts to copy the file given by \texttt{srcPathPtr} to the path name given by \texttt{destPathPtr}. If the two paths given lie in the same filesystem (according to \texttt{Tcl\_FSGetFileSystemForPath}) then that filesystem's 'copy file' function is called (if it is non−NULL). Otherwise the function returns −1 and sets Tcl's errno to the 'EXDEV' posix error code (which signifies a 'cross−domain link').

\texttt{Tcl\_FSCopyDirectory} attempts to copy the directory given by \texttt{srcPathPtr} to the path name given by \texttt{destPathPtr}. If the two paths given lie in the same filesystem (according to \texttt{Tcl\_FSGetFileSystemForPath})
then that filesystem's 'copy file' function is called (if it is non-NULL). Otherwise the function returns −1 and sets Tcl's errno to the 'EXDEV' posix error code (which signifies a 'cross-domain link').

Tcl_FSCreateDirectory attempts to create the directory given by pathPtr by calling the owning filesystem's 'create directory' function.

Tcl_FSDeleteFile attempts to delete the file given by pathPtr by calling the owning filesystem's 'delete file' function.

Tcl_FSRemoveDirectory attempts to remove the directory given by pathPtr by calling the owning filesystem's 'remove directory' function.

Tcl_FSRenameFile attempts to rename the file or directory given by srcPathPtr to the path name given by destPathPtr. If the two paths given lie in the same filesystem (according to Tcl_FSGetFileSystemForPath) then that filesystem's 'rename file' function is called (if it is non-NULL). Otherwise the function returns −1 and sets Tcl's errno to the 'EXDEV' posix error code (which signifies a 'cross-domain link').

Tcl_FSListVolumes calls each filesystem which has a non-NULL 'list volumes' function and asks them to return their list of root volumes. It accumulates the return values in a list which is returned to the caller (with a refCount of 0).

Tcl_FSEvalFile reads the file given by pathPtr and evaluates its contents as a Tcl script. It returns the same information as Tcl_EvalObjEx. If the file couldn't be read then a Tcl error is returned to describe why the file couldn't be read. The eofchar for files is '\32' (^Z) for all platforms. If you require a ``^Z'' in code for string comparison, you can use ``\032'' or ``\u001a'', which will be safely substituted by the Tcl interpreter into ``^Z''.

Tcl_FSLoadFile dynamically loads a binary code file into memory and returns the addresses of two procedures within that file, if they are defined. The appropriate function for the filesystem to which pathPtr belongs will be called. If that filesystem does not implement this function (most virtual filesystems will not, because of OS limitations in dynamically loading binary code), Tcl will attempt to copy the file to a temporary directory and load that temporary file.

Returns a standard Tcl completion code. If an error occurs, an error message is left in the interp's result.

Tcl_FSMatchInDirectory is used by the globbing code to search a directory for all files which match a given pattern. The appropriate function for the filesystem to which pathPtr belongs will be called.

The return value is a standard Tcl result indicating whether an error occurred in globbing. Error messages are placed in interp, but good results are placed in the resultPtr given. Note that the 'glob' code implements recursive patterns internally, so this function will only ever be passed simple patterns, which can be matched using the logic of 'string match'. To handle recursion, Tcl will call this function frequently asking only for directories to be returned.

Tcl_FSLink replaces the library version of readlink(), and extends it to support the creation of links. The
appropriate function for the filesystem to which linkNamePtr belongs will be called.

If the toPtr is NULL, a readlink action is performed. The result is a Tcl_Obj specifying the contents of the symbolic link given by linkNamePtr, or NULL if the link could not be read. The result is owned by the caller, which should call Tcl_DecrRefCount when the result is no longer needed. If the toPtr is not NULL, Tcl should create a link of one of the types passed in in the linkAction flag. This flag is an or'd combination of TCL_CREATE_SYMBOLIC_LINK and TCL_CREATE_HARD_LINK. Where a choice exists (i.e. more than one flag is passed in), the Tcl convention is to prefer symbolic links. When a link is successfully created, the return value should be toPtr (which is therefore already owned by the caller). If unsuccessful, NULL should be returned.

Tcl_FSLstat fills the stat structure statPtr with information about the specified file. You do not need any access rights to the file to get this information but you need search rights to all directories named in the path leading to the file. The stat structure includes info regarding device, inode (always 0 on Windows), privilege mode, nlink (always 1 on Windows), user id (always 0 on Windows), group id (always 0 on Windows), rdev (same as device on Windows), size, last access time, last modification time, and creation time.

If path exists, Tcl_FSLstat returns 0 and the stat structure is filled with data. Otherwise, −1 is returned, and no stat info is given.

Tcl_FSUtime replaces the library version of utime.

For results see 'utime' documentation. If successful, the function will update the 'atime' and 'mtime' values of the file given.

Tcl_FSFileAttrsGet implements read access for the hookable 'file attributes' subcommand. The appropriate function for the filesystem to which pathPtr belongs will be called.

If the result is TCL_OK, then an object was placed in objPtrRef, which will only be temporarily valid (unless Tcl_IncrRefCount is called).

Tcl_FSFileAttrsSet implements write access for the hookable 'file attributes' subcommand. The appropriate function for the filesystem to which pathPtr belongs will be called.

Tcl_FSFileAttrStrings implements part of the hookable 'file attributes' subcommand. The appropriate function for the filesystem to which pathPtr belongs will be called.

The called procedure may either return an array of strings, or may instead return NULL and place a Tcl list into the given objPtrRef. Tcl will take that list and first increment its refCount before using it. On completion of that use, Tcl will decrement its refCount. Hence if the list should be disposed of by Tcl when done, it should have a refCount of zero, and if the list should not be disposed of, the filesystem should ensure it retains a refCount on the object.

Tcl_FSAccess checks whether the process would be allowed to read, write or test for existence of the file (or other file system object) whose name is pathname. If pathname is a symbolic link on Unix, then permissions
of the file referred by this symbolic link are tested.

On success (all requested permissions granted), zero is returned. On error (at least one bit in mode asked for a permission that is denied, or some other error occurred), −1 is returned.

Tcl_FSStat fills the stat structure statPtr with information about the specified file. You do not need any access rights to the file to get this information but you need search rights to all directories named in the path leading to the file. The stat structure includes info regarding device, inode (always 0 on Windows), privilege mode, nlink (always 1 on Windows), user id (always 0 on Windows), group id (always 0 on Windows), rdev (same as device on Windows), size, last access time, last modification time, and creation time.

If path exists, Tcl_FSStat returns 0 and the stat structure is filled with data. Otherwise, −1 is returned, and no stat info is given.

Tcl_FSOpenFileChannel opens a file specified by pathPtr and returns a channel handle that can be used to perform input and output on the file. This API is modeled after the fopen procedure of the Unix standard I/O library. The syntax and meaning of all arguments is similar to those given in the Tcl open command when opening a file. If an error occurs while opening the channel, Tcl_FSOpenFileChannel returns NULL and records a POSIX error code that can be retrieved with Tcl_GetErrno. In addition, if interp is non-NULL, Tcl_FSOpenFileChannel leaves an error message in interp's result after any error.

The newly created channel is not registered in the supplied interpreter; to register it, use Tcl_RegisterChannel, described below. If one of the standard channels, stdin, stdout or stderr was previously closed, the act of creating the new channel also assigns it as a replacement for the standard channel.

Tcl_FSGetCwd replaces the library version of getcwd().

It returns the Tcl library's current working directory. This may be different to the native platform's working directory, in the case for which the cwd is not in the native filesystem.

The result is a pointer to a Tcl_Obj specifying the current directory, or NULL if the current directory could not be determined. If NULL is returned, an error message is left in the interp's result. The result already has its refCount incremented for the caller. When it is no longer needed, that refCount should be decremented. This is needed for thread-safety purposes, to allow multiple threads to access this and related functions, while ensuring the results are always valid.

Tcl_FSChdir replaces the library version of chdir(). The path is normalized and then passed to the filesystem which claims it. If that filesystem does not implement this function, Tcl will fallback to a combination of stat and access to check whether the directory exists and has appropriate permissions.

For results, see chdir() documentation. If successful, we keep a record of the successful path in cwdPathPtr for subsequent calls to getcwd.

Tcl_FSPathSeparator returns the separator character to be used for most specific element of the path
specified by pathPtr (i.e. the last part of the path).

The separator is returned as a Tcl_Obj containing a string of length 1. If the path is invalid, NULL is returned.

**Tcl_FSJoinPath** takes the given Tcl_Obj, which should be a valid list, and returns the path object given by considering the first 'elements' elements as valid path segments. If elements < 0, we use the entire list.

Returns object with refCount of zero, containing the joined path.

**Tcl_FSSplitPath** takes the given Tcl_Obj, which should be a valid path, and returns a Tcl List object containing each segment of that path as an element.

Returns list object with refCount of zero. If the passed in lenPtr is non–NULL, we use it to return the number of elements in the returned list.

**Tcl_FSEqualPaths** tests whether the two paths given represent the same filesystem object

It returns 1 if the paths are equal, and 0 if they are different. If either path is NULL, 0 is always returned.

**Tcl_FSGetNormalizedPath** this important function attempts to extract from the given Tcl_Obj a unique normalized path representation, whose string value can be used as a unique identifier for the file.

It returns the normalized path object, with refCount of zero, or NULL if the path was invalid or could otherwise not be successfully converted. Extraction of absolute, normalized paths is very efficient (because the filesystem operates on these representations internally), although the result when the filesystem contains numerous symbolic links may not be the most user–friendly version of a path.

**Tcl_FSJoinToPath** takes the given object, which should usually be a valid path or NULL, and joins onto it the array of paths segments given.

Returns object with refCount of zero, containing the joined path.

**Tcl_FSCreateToPathType** tries to convert the given Tcl_Obj to a valid Tcl path type, taking account of the fact that the cwd may have changed even if this object is already supposedly of the correct type. The filename may begin with ”~” (to indicate current user's home directory) or ”~<user>" (to indicate any user's home directory).

If the conversion succeeds (i.e. the object is a valid path in one of the current filesystems), then TCL_OK is returned. Otherwise TCL_ERROR is returned, and an error message may be left in the interpreter.

**Tcl_FSGetInternalRep** extracts the internal representation of a given path object, in the given filesystem. If the path object belongs to a different filesystem, we return NULL. If the internal representation is currently NULL, we attempt to generate it, by calling the filesystem's **Tcl_FSCreateInternalRepProc**.

Returns NULL or a valid internal path representation. This internal representation is cached, so that repeated
calls to this function will not require additional conversions.

**Tcl_FSGetTranslatedPath** attempts to extract the translated path from the given Tcl_Obj.

If the translation succeeds (i.e. the object is a valid path), then it is returned. Otherwise NULL will be returned, and an error message may be left in the interpreter. A "translated" path is one which contains no "~" or "~user" sequences (these have been expanded to their current representation in the filesystem). The object returned is owned by the caller, which must store it or call Tcl_DecrRefCount to ensure memory is freed. This function is of little practical use, and **Tcl_FSGetNormalizedPath** or **Tcl_GetNativePath** are usually better functions to use for most purposes.

**Tcl_FSGetTranslatedStringPath** does the same as **Tcl_FSGetTranslatedPath**, but returns a character string or NULL. The string returned is dynamically allocated and owned by the caller, which must store it or call ckfree to ensure it is freed. Again, **Tcl_FSGetNormalizedPath** or **Tcl_GetNativePath** are usually better functions to use for most purposes.

**Tcl_FSNewNativePath** performs something like that reverse of the usual obj->path->nativerep conversions. If some code retrieves a path in native form (from, e.g. readlink or a native dialog), and that path is to be used at the Tcl level, then calling this function is an efficient way of creating the appropriate path object type.

The resulting object is a pure 'path' object, which will only receive a Utf–8 string representation if that is required by some Tcl code.

**Tcl_FSGetNativePath** is for use by the Win/Unix/MacOS native filesystems, so that they can easily retrieve the native (char* or TCHAR*) representation of a path. This function is a convenience wrapper around **Tcl_FSGetInternalRep**, and assumes the native representation is string-based. It may be desirable in the future to have non-string–based native representations (for example, on MacOS, a representation using a fileSpec of FSRef structure would probably be more efficient). On Windows a full Unicode representation would allow for paths of unlimited length. Currently the representation is simply a character string containing the complete, absolute path in the native encoding.

The native representation is cached so that repeated calls to this function will not require additional conversions.

**Tcl_FSFileSystemInfo** returns a list of two elements. The first element is the name of the filesystem (e.g. "native" or "vfs" or "zip" or "prowrap", perhaps), and the second is the particular type of the given path within that filesystem (which is filesystem dependent). The second element may be empty if the filesystem does not provide a further categorization of files.

A valid list object is returned, unless the path object is not recognized, when NULL will be returned.

**Tcl_FSGetFileSystemForPath** returns the a pointer to the **Tcl_Filesystem** which accepts this path as valid.

If no filesystem will accept the path, NULL is returned.
**Tcl_FSGetPathType** determines whether the given path is relative to the current directory, relative to the current volume, or absolute.

It returns one of **TCL_PATH_ABSOLUTE**, **TCL_PATH_RELATIVE**, or **TCL_PATH_VOLUME_RELATIVE**.

**Tcl_AllocStatBuf** allocates a **Tcl_StatBuf** on the system heap (which may be deallocated by being passed to **ckfree**) This allows extensions to invoke **Tcl_FSStat** and **Tcl_FSLStat** without being dependent on the size of the buffer. That in turn depends on the flags used to build Tcl.

**TCL_FILESYSTEM**

A filesystem provides a **Tcl_Filesystem** structure that contains pointers to functions that implement the various operations on a filesystem; these operations are invoked as needed by the generic layer, which generally occurs through the functions listed above.

The **Tcl_Filesystem** structures are manipulated using the following methods.

**Tcl_FSRegister** takes a pointer to a filesystem structure and an optional piece of data to associated with that filesystem. On calling this function, Tcl will attach the filesystem to the list of known filesystems, and it will become fully functional immediately. Tcl does not check if the same filesystem is registered multiple times (and in general that is not a good thing to do). **TCL_OK** will be returned.

**Tcl_FSUnregister** removes the given filesystem structure from the list of known filesystems, if it is known, and returns **TCL_OK**. If the filesystem is not currently registered, **TCL_ERROR** is returned.

**Tcl_FSData** will return the ClientData associated with the given filesystem, if that filesystem is registered. Otherwise it will return NULL.

**Tcl_FSMountsChanged** is used to inform the Tcl's core that the set of mount points for the given (already registered) filesystem have changed, and that cached file representations may therefore no longer be correct.

The **Tcl_Filesystem** structure contains the following fields:

```c
typedef struct Tcl_Filesystem {
    CONST char *typeName;
    int structureLength;
    Tcl_FSVersion version;
    Tcl_FSPathInFilesystemProc *pathInFilesystemProc;
    Tcl_FSDupInternalRepProc *dupInternalRepProc;
    Tcl_FSFreeInternalRepProc *freeInternalRepProc;
    Tcl_FSInternalToNormalizedProc *internalToNormalizedProc;
    Tcl_FSCreateInternalRepProc *createInternalRepProc;
    Tcl_FSNormalizePathProc *normalizePathProc;
    Tcl_FSFilesystemPathTypeProc *filesystemPathTypeProc;
    Tcl_FSFilesystemSeparatorProc *filesystemSeparatorProc;
    Tcl_FSStatProc *statProc;
    Tcl_FSAccessProc *accessProc;
}
```
Except for the first three fields in this structure which contain simple data elements, all entries contain addresses of functions called by the generic filesystem layer to perform the complete range of filesystem related actions.

The many functions in this structure are broken down into three categories: infrastructure functions (almost all of which must be implemented), operational functions (which must be implemented if a complete filesystem is provided), and efficiency functions (which need only be implemented if they can be done so efficiently, or if they have side−effects which are required by the filesystem; Tcl has less efficient emulations it can fall back on). It is important to note that, in the current version of Tcl, most of these fallbacks are only used to handle commands initiated in Tcl, not in C. What this means is, that if a 'file rename' command is issued in Tcl, and the relevant filesystem(s) do not implement their Tcl_FSRenameFileProc, Tcl's core will instead fallback on a combination of other filesystem functions (it will use Tcl_FSCopyFileProc followed by Tcl_FSDeleteFileProc, and if Tcl_FSCopyFileProc is not implemented there is a further fallback). However, if a Tcl_FSRenameFile command is issued at the C level, no such fallbacks occur. This is true except for the last four entries in the filesystem table (Istat, load, getcwd and chdir) for which fallbacks do in fact occur at the C level.

As an example, here is the filesystem lookup table used by the "vfs" extension which allows filesystem actions to be implemented in Tcl.

static Tcl_Filesystem vfsFileSystem = {
"tclvfs",
sizeof(Tcl_Filesystem),
TCL_FILESYSTEM_VERSION_1,
&VfsPathInFilesystem,
&VfsDupInternalRep,
&VfsFreeInternalRep,
/* No internal to normalized, since we don't create any
 * pure 'internal' Tcl_Obj path representations */
NULL,
/* No create native rep function, since we don't use it
Any functions which take path names in Tcl_Obj form take those names in UTF−8 form. The filesystem infrastructure API is designed to support efficient, cached conversion of these UTF−8 paths to other native representations.

**TYPENAME**

The *typeName* field contains a null−terminated string that identifies the type of the filesystem implemented, e.g. native or zip or vfs.

**STRUCTURE LENGTH**

The *structureLength* field is generally implemented as sizeof(Tcl_Filesystem), and is there to allow easier binary backwards compatibility if the size of the structure changes in a future Tcl release.
VERSION

The version field should be set to TCL_FILESYSTEM_VERSION_1.

FILESYSTEM INFRASTRUCTURE

These fields contain addresses of functions which are used to associate a particular filesystem with a file path, and deal with the internal handling of path representations, for example copying and freeing such representations.

PATHINFILESYSTEMPROC

The pathInFilesystemProc field contains the address of a function which is called to determine whether a given path object belongs to this filesystem or not. Tcl will only call the rest of the filesystem functions with a path for which this function has returned TCL_OK. If the path does not belong, −1 should be returned (the behaviour of Tcl for any other return value is not defined). If TCL_OK is returned, then the optional clientDataPtr output parameter can be used to return an internal (filesystem specific) representation of the path, which will be cached inside the path object, and may be retrieved efficiently by the other filesystem functions. Tcl will simultaneously cache the fact that this path belongs to this filesystem. Such caches are invalidated when filesystem structures are added or removed from Tcl's internal list of known filesystems.

typedef int Tcl_FSPathInFilesystemProc(
    Tcl_Obj * pathPtr,
    ClientData * clientDataPtr);

DUPINTERNALREPPROC

This function makes a copy of a path's internal representation, and is called when Tcl needs to duplicate a path object. If NULL, Tcl will simply not copy the internal representation, which may then need to be regenerated later.

typedef ClientData Tcl_FSDupInternalRepProc(
    ClientData clientData);

FREEINTERNALREPPROC

Free the internal representation. This must be implemented if internal representations need freeing (i.e. if some memory is allocated when an internal representation is generated), but may otherwise be NULL.

typedef void Tcl_FSFreeInternalRepProc(
    ClientData clientData);

INTERNALTONORMALIZEDPROC

Function to convert internal representation to a normalized path. Only required if the filesystem creates pure
path objects with no string/path representation. The return value is a Tcl object whose string representation is the normalized path.

typedef Tcl_Obj* Tcl_FSInternalToNormalizedProc(
    ClientData clientData);

CREATEINTERNALREPPROC

Function to take a path object, and calculate an internal representation for it, and store that native representation in the object. May be NULL if paths have no internal representation, or if the Tcl_FSPathInFileSystemProc for this filesystem always immediately creates an internal representation for paths it accepts.

typedef ClientData Tcl_FSCreateInternalRepProc(
    Tcl_Obj *pathPtr);

NORMALIZEPATHPROC

Function to normalize a path. Should be implemented for all filesystems which can have multiple string representations for the same path object. In Tcl, every 'path' must have a single unique 'normalized' string representation. Depending on the filesystem, there may be more than one unnormalized string representation which refers to that path (e.g. a relative path, a path with different character case if the filesystem is case insensitive, a path contain a reference to a home directory such as '~', a path containing symbolic links, etc). If the very last component in the path is a symbolic link, it should not be converted into the object it points to (but its case or other aspects should be made unique). All other path components should be converted from symbolic links. This one exception is required to agree with Tcl's semantics with 'file delete', 'file rename', 'file copy' operating on symbolic links. This function may be called with 'nextCheckpoint' either at the beginning of the path (i.e. zero), at the end of the path, or at any intermediate file separator in the path. It will never point to any other arbitrary position in the path. In the last of the three valid cases, the implementation can assume that the path up to and including the file separator is known and normalized.

typedef int Tcl_FSNormalizePathProc(
    Tcl_Interp *interp,
    Tcl_Obj *pathPtr,
    int nextCheckpoint);

FILESYSTEM OPERATIONS

The fields in this section of the structure contain addresses of functions which are called to carry out the basic filesystem operations. A filesystem which expects to be used with the complete standard Tcl command set must implement all of these. If some of them are not implemented, then certain Tcl commands may fail when operating on paths within that filesystem. However, in some instances this may be desirable (for example, a read-only filesystem should not implement the last four functions, and a filesystem which does not support symbolic links need not implement the readlink function, etc. The Tcl core expects filesystems to behave in this way).
**FILESYSTEMPATHTYPEPROC**

Function to determine the type of a path in this filesystem. May be NULL, in which case no type information will be available to users of the filesystem. The 'type' is used only for informational purposes, and should be returned as the string representation of the Tcl_Obj which is returned. A typical return value might be "networked", "zip" or "ftp". The Tcl_Obj result is owned by the filesystem and so Tcl will increment the refCount of that object if it wishes to retain a reference to it.

```c
typedef Tcl_Obj* Tcl_FSFilesystemPathTypeProc(
    Tcl_Obj * pathPtr);
```

**FILESYSTEMSEPARATORPROC**

Function to return the separator character(s) for this filesystem. Must be implemented, otherwise the file separator command will not function correctly. The usual return value will be a Tcl_Obj containing the string "/".

```c
typedef Tcl_Obj* Tcl_FSFilesystemSeparatorProc(
    Tcl_Obj * pathPtr);
```

**STATPROC**

Function to process a Tcl_FSStat() call. Must be implemented for any reasonable filesystem, since many Tcl level commands depend crucially upon it (e.g. file atime, file isdirectory, file size, glob).

```c
typedef int Tcl_FSStatProc(
    Tcl_Obj * pathPtr,
    Tcl_StatBuf * statPtr);
```

The Tcl_FSStatProc fills the stat structure statPtr with information about the specified file. You do not need any access rights to the file to get this information but you need search rights to all directories named in the path leading to the file. The stat structure includes info regarding device, inode (always 0 on Windows), privilege mode, nlink (always 1 on Windows), user id (always 0 on Windows), group id (always 0 on Windows), rdev (same as device on Windows), size, last access time, last modification time, and creation time.

If the file represented by pathPtr exists, the Tcl_FSStatProc returns 0 and the stat structure is filled with data. Otherwise, −1 is returned, and no stat info is given.

**ACCESSPROC**

Function to process a Tcl_FSAccess() call. Must be implemented for any reasonable filesystem, since many Tcl level commands depend crucially upon it (e.g. file exists, file readable).

```c
typedef int Tcl_FSAccessProc(
    Tcl_Obj * pathPtr);
```
mode int

The **Tcl_FSAccessProc** checks whether the process would be allowed to read, write or test for existence of the file (or other file system object) whose name is pathname. If pathname is a symbolic link, then permissions of the file referred by this symbolic link should be tested.

On success (all requested permissions granted), zero is returned. On error (at least one bit in mode asked for a permission that is denied, or some other error occurred), −1 is returned.

**OPENFILECHANNELPROC**

Function to process a **Tcl_FSOpenFileChannel()** call. Must be implemented for any reasonable filesystem, since any operations which require open or accessing a file's contents will use it (e.g. **open**, **encoding**, and many Tk commands).

```c
typedef Tcl_Channel Tcl_FSOpenFileChannelProc(
    Tcl_Interp *interp,
    Tcl_Obj *pathPtr,
    int mode,
    int permissions);
```

The **Tcl_FSOpenFileChannelProc** opens a file specified by **pathPtr** and returns a channel handle that can be used to perform input and output on the file. This API is modeled after the **fopen** procedure of the Unix standard I/O library. The syntax and meaning of all arguments is similar to those given in the Tcl **open** command when opening a file, where the **mode** argument is a combination of the POSIX flags O_RDONLY, O_WRONLY, etc. If an error occurs while opening the channel, the **Tcl_FSOpenFileChannelProc** returns NULL and records a POSIX error code that can be retrieved with **Tcl_GetErrno**. In addition, if **interp** is non−NULL, the **Tcl_FSOpenFileChannelProc** leaves an error message in **interp**'s result after any error.

The newly created channel is not registered in the supplied interpreter; to register it, use **Tcl_RegisterChannel**. If one of the standard channels, **stdin**, **stdout** or **stderr** was previously closed, the act of creating the new channel also assigns it as a replacement for the standard channel.

**MATCHINDIRECTORYPROC**

Function to process a **Tcl_FSMatchInDirectory()** call. If not implemented, then glob and recursive copy functionality will be lacking in the filesystem (and this may impact commands like 'encoding names' which use glob functionality internally).

```c
typedef int Tcl_FSMatchInDirectoryProc(
    Tcl_Interp* interp,
    Tcl_Obj *result,
    Tcl_Obj *pathPtr,
    CONST char *pattern,
    Tcl_GlobTypeData *types);
```

OPENFILECHANNELPROC 722
The function should return all files or directories (or other filesystem objects) which match the given pattern and accord with the types specification given. There are two ways in which this function may be called. If pattern is NULL, then pathPtr is a full path specification of a single file or directory which should be checked for existence and correct type. Otherwise, pathPtr is a directory, the contents of which the function should search for files or directories which have the correct type. In either case, pathPtr can be assumed to be both non–NULL and non–empty. It is not currently documented whether pathPtr will have a file separator at its end of not, so code should be flexible to both possibilities.

The return value is a standard Tcl result indicating whether an error occurred in the matching process. Error messages are placed in interp, but on a TCL_OK result, the interpreter should not be modified, but rather results should be added to the result object given (which can be assumed to be a valid Tcl list). The matches added to result should include any path prefix given in pathPtr (this usually means they will be absolute path specifications). Note that if no matches are found, that simply leads to an empty result — errors are only signaled for actual file or filesystem problems which may occur during the matching process.

**UTIMEPROC**

Function to process a Tcl_FSUtime() call. Required to allow setting (not reading) of times with 'file mtime', 'file atime' and the open–r/open–w/fcopy implementation of 'file copy'.

```c
typedef int Tcl_FSUtimeProc(
    Tcl_Obj *pathPtr,
    struct utimbuf *tval
);
```

The access and modification times of the file specified by pathPtr should be changed to the values given in the tval structure.

The return value is a standard Tcl result indicating whether an error occurred in the process.

**LINKPROC**

Function to process a Tcl_FSLink() call. Should be implemented only if the filesystem supports links, and may otherwise be NULL.

```c
typedef Tcl_Obj* Tcl_FSLinkProc(
    Tcl_Obj *linkNamePtr,
    Tcl_Obj *toPtr,
    int linkAction
);
```

If toPtr is NULL, the function is being asked to read the contents of a link. The result is a Tcl_Obj specifying the contents of the link given by linkNamePtr, or NULL if the link could not be read. The result is owned by the caller, which should call Tcl_DecrRefCount when the result is no longer needed. If toPtr is not NULL, the function should attempt to create a link. The result in this case should be toPtr if the link was successful and NULL otherwise. In this case the result is not owned by the caller. See the documentation for Tcl_FSLink for the correct interpretation of the linkAction flags.
LISTVOLUMESPROC

Function to list any filesystem volumes added by this filesystem. Should be implemented only if the filesystem adds volumes at the head of the filesystem, so that they can be returned by 'file volumes'.

```c
typedef Tcl_Obj* Tcl_FSListVolumesProc(void);
```

The result should be a list of volumes added by this filesystem, or NULL (or an empty list) if no volumes are provided. The result object is considered to be owned by the filesystem (not by Tcl’s core), but should be given a refCount for Tcl. Tcl will use the contents of the list and then decrement that refCount. This allows filesystems to choose whether they actually want to retain a ‘master list' of volumes or not (if not, they generate the list on the fly and pass it to Tcl with a refCount of 1 and then forget about the list, if yes, then they simply increment the refCount of their master list and pass it to Tcl which will copy the contents and then decrement the count back to where it was).

Therefore, Tcl considers return values from this proc to be read-only.

FILEATTRSTRINGSPROC

Function to list all attribute strings which are valid for this filesystem. If not implemented the filesystem will not support the `file attributes' command. This allows arbitrary additional information to be attached to files in the filesystem. If it is not implemented, there is no need to implement the `get' and `set' methods.

```c
typedef CONST char** Tcl_FSFileAttrStringsProc(
    Tcl_Obj *pathPtr,
    Tcl_Obj** objPtrRef);
```

The called function may either return an array of strings, or may instead return NULL and place a Tcl list into the given objPtrRef. Tcl will take that list and first increment its refCount before using it. On completion of that use, Tcl will decrement its refCount. Hence if the list should be disposed of by Tcl when done, it should have a refCount of zero, and if the list should not be disposed of, the filesystem should ensure it retains a refCount on the object.

FILEATTRSGETPROC

Function to process a `Tcl_FSFileAttrsGet()' call, used by `file attributes'.

```c
typedef int Tcl_FSFileAttrsGetProc(
    Tcl_Interp *interp,
    int index,
    Tcl_Obj *pathPtr,
    Tcl_Obj **objPtrRef);
```

Returns a standard Tcl return code. The attribute value retrieved, which corresponds to the `index'th element in the list returned by the Tcl_FSFileAttrStringsProc, is a Tcl_Obj placed in objPtrRef (if TCL_OK was returned) and is likely to have a refCount of zero. Either way we must either store it somewhere (e.g. the Tcl
Result), or Incr/Decr its refCount to ensure it is properly freed.

**FILEATTRSSETPROC**

Function to process a `Tcl_FSFileAttrsSet()` call, used by 'file attributes'. If the filesystem is read–only, there is no need to implement this.

```c
typedef int Tcl_FSFileAttrsSetProc(
    Tcl_Interp *interp,
    int index,
    Tcl_Obj *pathPtr,
    Tcl_Obj *objPtr);
```

The attribute value of the `index`'th element in the list returned by the `Tcl_FSFileAttrStringsProc` should be set to the `objPtr` given.

**CREATEDIRECTORYPROC**

Function to process a `Tcl_FSCreateDirectory()` call. Should be implemented unless the FS is read–only.

```c
typedef int Tcl_FSCreateDirectoryProc(
    Tcl_Obj *pathPtr);
```

The return value is a standard Tcl result indicating whether an error occurred in the process. If successful, a new directory should have been added to the filesystem in the location specified by `pathPtr`.

**REMOVEDIRECTORYPROC**

Function to process a `Tcl_FSRemoveDirectory()` call. Should be implemented unless the FS is read–only.

```c
typedef int Tcl_FSRemoveDirectoryProc(
    Tcl_Obj *pathPtr,
    int recursive,
    Tcl_Obj **errorPtr);
```

The return value is a standard Tcl result indicating whether an error occurred in the process. If successful, the directory specified by `pathPtr` should have been removed from the filesystem. If the `recursive` flag is given, then a non–empty directory should be deleted without error. If an error does occur, the name of the file or directory which caused the error should be placed in `errorPtr`.

**DELETEFILEPROC**

Function to process a `Tcl_FSDeleteFile()` call. Should be implemented unless the FS is read–only.

```c
typedef int Tcl_FSDeleteFileProc(
    Tcl_Obj *pathPtr);
```
The return value is a standard Tcl result indicating whether an error occurred in the process. If successful, the file specified by pathPtr should have been removed from the filesystem. Note that, if the filesystem supports symbolic links, Tcl will always call this function and not Tcl_FSRemoveDirectoryProc when needed to delete them (even if they are symbolic links to directories).

FILESYSTEM EFFICIENCY

LSTATPROC

Function to process a Tcl_FSLstat() call. If not implemented, Tcl will attempt to use the statProc defined above instead. Therefore it need only be implemented if a filesystem can differentiate between stat and lstat calls.

typedef int Tcl_FSLstatProc(
    Tcl_Obj * pathPtr,
    Tcl_StatBuf * statPtr);

The behavior of this function is very similar to that of the Tcl_FSStatProc defined above, except that if it is applied to a symbolic link, it returns information about the link, not about the target file.

COPYFILEPROC

Function to process a Tcl_FSCopyFile() call. If not implemented Tcl will fall back on open−r, open−w and fcopy as a copying mechanism. Therefore it need only be implemented if the filesystem can perform that action more efficiently.

typedef int Tcl_FSCopyFileProc(
    Tcl_Obj * srcPathPtr,
    Tcl_Obj * destPathPtr);

The return value is a standard Tcl result indicating whether an error occurred in the copying process. Note that, destPathPtr is the name of the file which should become the copy of srcPathPtr. It is never the name of a directory into which srcPathPtr could be copied (i.e. the function is much simpler than the Tcl level 'file copy' subcommand). Note that, if the filesystem supports symbolic links, Tcl will always call this function and not Tcl_FSCopyDirectoryProc when needed to copy them (even if they are symbolic links to directories).

RENAMEFILEPROC

Function to process a Tcl_FSRenameFile() call. If not implemented, Tcl will fall back on a copy and delete mechanism. Therefore it need only be implemented if the filesystem can perform that action more efficiently.

typedef int Tcl_FSRenameFileProc(
    Tcl_Obj * srcPathPtr,
    Tcl_Obj * destPathPtr);

The return value is a standard Tcl result indicating whether an error occurred in the renaming process.
COPYDIRECTORYPROC

Function to process a Tcl_FSCopyDirectory() call. If not implemented, Tcl will fall back on a recursive create−dir, file copy mechanism. Therefore it need only be implemented if the filesystem can perform that action more efficiently.

typedef int Tcl_FSCopyDirectoryProc(
    Tcl_Obj *srcPathPtr,
    Tcl_Obj *destPathPtr,
    Tcl_Obj **errorPtr);

The return value is a standard Tcl result indicating whether an error occurred in the copying process. If an error does occur, the name of the file or directory which caused the error should be placed in errorPtr. Note that, destPathPtr is the name of the directory−name which should become the mirror−image of srcPathPtr. It is not the name of a directory into which srcPathPtr should be copied (i.e. the function is much simpler than the Tcl level 'file copy' subcommand).

LOADFILEPROC

Function to process a Tcl_FSLoadFile() call. If not implemented, Tcl will fall back on a copy to native−temp followed by a Tcl_FSLoadFile on that temporary copy. Therefore it need only be implemented if the filesystem can load code directly, or it can be implemented simply to return TCL_ERROR to disable load functionality in this filesystem entirely.

typedef int Tcl_FSLoadFileProc(
    Tcl_Interp *interp,
    Tcl_Obj *pathPtr,
    Tcl_LoadHandle *handlePtr,
    Tcl_FSUnloadFileProc *unloadProcPtr);

Returns a standard Tcl completion code. If an error occurs, an error message is left in the interp's result. The function dynamically loads a binary code file into memory. On a successful load, the handlePtr should be filled with a token for the dynamically loaded file, and the unloadProcPtr should be filled in with the address of a procedure. The procedure will be called with the given Tcl_LoadHandle as its only parameter when Tcl needs to unload the file.

UNLOADFILEPROC

Function to unload a previously successfully loaded file. If load was implemented, then this should also be implemented, if there is any cleanup action required.

typedef void Tcl_FSUnloadFileProc(
    Tcl_LoadHandlehandle);
GETCWDPROC

Function to process a `Tcl_FSGetCwd()` call. Most filesystems need not implement this. It will usually only be called once, if 'getcwd' is called before 'chdir'. May be NULL.

```c
typedef Tcl_Obj* Tcl_FSGetCwdProc(
    Tcl_Interp * interp);
```

If the filesystem supports a native notion of a current working directory (which might perhaps change independent of Tcl), this function should return that cwd as the result, or NULL if the current directory could not be determined (e.g. the user does not have appropriate permissions on the cwd directory). If NULL is returned, an error message is left in the interp's result.

CHDIRPROC

Function to process a `Tcl_FSChdir()` call. If filesystems do not implement this, it will be emulated by a series of directory access checks. Otherwise, virtual filesystems which do implement it need only respond with a positive return result if the dirName is a valid, accessible directory in their filesystem. They need not remember the result, since that will be automatically remembered for use by GetCwd. Real filesystems should carry out the correct action (i.e. call the correct system 'chdir' api).

```c
typedef int Tcl_FSChdirProc(
    Tcl_Obj * pathPtr);
```

The `Tcl_FSChdirProc` changes the applications current working directory to the value specified in `pathPtr`. The function returns −1 on error or 0 on success.

KEYWORDS

stat access filesystem vfs

Copyright ' 2001 Vincent Darley
Copyright ' 1995-1997 Roger E. Critchlow Jr.
Notifier

NAME

Tcl_CreateEventSource, Tcl_DeleteEventSource, Tcl_SetMaxBlockTime, Tcl_QueueEvent,
Tcl_ThreadQueueEvent, Tcl_ThreadAlert, Tcl_GetCurrentThread, Tcl_DeleteEvents,
Tcl_InitNotifier, Tcl_FinalizeNotifier, Tcl_WaitForEvent, Tcl_AlertNotifier, Tcl_SetTimer,
Tcl_ServiceAll, Tcl_ServiceEvent, Tcl_GetServiceMode, Tcl_SetServiceMode -- the event
queue and notifier interfaces

SYNOPSIS

ARGUMENTS
INTRODUCTION
NOTIFIER BASICS
CREATING A NEW EVENT SOURCE
   TCL_QUEUE_TAIL
   TCL_QUEUE_HEAD
   TCL_QUEUE_MARK
CREATING A NEW NOTIFIER
EXTERNAL EVENT LOOPS
SEE ALSO
KEYWORDS

NAME

Tcl_CreateEventSource, Tcl_DeleteEventSource, Tcl_SetMaxBlockTime, Tcl_QueueEvent,
Tcl_ThreadQueueEvent, Tcl_ThreadAlert, Tcl_GetCurrentThread, Tcl_DeleteEvents, Tcl_InitNotifier,
Tcl_FinalizeNotifier, Tcl_WaitForEvent, Tcl_AlertNotifier, Tcl_SetTimer, Tcl_ServiceAll,
Tcl_ServiceEvent, Tcl_GetServiceMode, Tcl_SetServiceMode -- the event queue and notifier interfaces

SYNOPSIS

#include <tcl.h>
void
Tcl_CreateEventSource(setupProc, checkProc, clientData)
void
Tcl_DeleteEventSource(setupProc, checkProc, clientData)
void
Tcl_SetMaxBlockTime(timePtr)
void
Tcl_QueueEvent(evPtr, position)
void
Tcl_ThreadQueueEvent(threadId, evPtr, position)
void
Tcl_ThreadAlert(threadId)
Tcl_ThreadId
Tcl_GetCurrentThread()
void
Tcl_DeleteEvents(deleteProc, clientData)
ClientData
Tcl_InitNotifier()
void
Tcl_FinalizeNotifier(clientData)
void
Tcl_WaitForEvent(timePtr)
void
Tcl_AlertNotifier(clientData)
void
Tcl_SetTimer(timePtr)
int
Tcl_ServiceAll()
int
Tcl_ServiceEvent(flags)
int
Tcl_GetServiceMode()
int
Tcl_SetServiceMode(mode)

ARGUMENTS

Tcl_EventSetupProc *setupProc (in)
Procedure to invoke to prepare for event wait in Tcl_DoOneEvent.

Tcl_EventCheckProc *checkProc (in)
Procedure for Tcl_DoOneEvent to invoke after waiting for events. Checks to see if any events have occurred and, if so, queues them.

ClientData clientData (in)
Arbitrary one-word value to pass to setupProc, checkProc, or deleteProc.

Tcl_Time *timePtr (in)
Indicates the maximum amount of time to wait for an event. This is specified as an interval (how long to wait), not an absolute time (when to wakeup). If the pointer passed to Tcl_WaitForEvent is NULL, it means there is no maximum wait time: wait forever if necessary.

Tcl_Event *evPtr (in)
An event to add to the event queue. The storage for the event must have been allocated by the caller using Tcl_Alloc or calloc.

Tcl_QueuePosition position (in)
Where to add the new event in the queue: TCL_QUEUE_TAIL, TCL_QUEUE_HEAD, or TCL_QUEUE_MARK.

Tcl_ThreadId threadId (in)
INTRODUCTION

The interfaces described here are used to customize the Tcl event loop. The two most common customizations are to add new sources of events and to merge Tcl's event loop with some other event loop, such as one provided by an application in which Tcl is embedded. Each of these tasks is described in a separate section below.

The procedures in this manual entry are the building blocks out of which the Tcl event notifier is constructed. The event notifier is the lowest layer in the Tcl event mechanism. It consists of three things:

[1] Event sources: these represent the ways in which events can be generated. For example, there is a timer event source that implements the `Tcl_CreateTimerHandler` procedure and the `after` command, and there is a file event source that implements the `Tcl_CreateFileHandler` procedure on Unix systems. An event source must work with the notifier to detect events at the right times, record them on the event queue, and eventually notify higher-level software that they have occurred. The procedures `Tcl_CreateEventSource`, `Tcl_DeleteEventSource`, and `Tcl_SetMaxBlockTime`, `Tcl_QueueEvent`, and `Tcl_DeleteEvents` are used primarily by event sources.

[2] The event queue: for non-threaded applications, there is a single queue for the whole application, containing events that have been detected but not yet serviced. Event sources place events onto the queue so that they may be processed in order at appropriate times during the event loop. The event queue guarantees a fair discipline of event handling, so that no event source can starve the others. It also allows events to be saved for servicing at a future time. Threaded applications work in a similar manner, except that there is a separate event queue for each thread containing a Tcl interpreter. `Tcl_QueueEvent` is used (primarily by event sources) to add events to the event queue and `Tcl_DeleteEvents` is used to remove events from the queue without processing them. In a threaded application, `Tcl_QueueEvent` adds an event to the current thread's queue, and `Tcl_ThreadQueueEvent` adds an event to a queue in a specific thread.

[3] The event loop: in order to detect and process events, the application enters a loop that waits for events to occur, places them on the event queue, and then processes them. Most applications will do this by calling the procedure `Tcl_DoOneEvent`, which is described in a separate manual entry.

Most Tcl applications need not worry about any of the internals of the Tcl notifier. However, the notifier now has enough flexibility to be retargeted either for a new platform or to use an external event loop (such as the...
Motif event loop, when Tcl is embedded in a Motif application). The procedures \texttt{Tcl\_WaitForEvent} and \texttt{Tcl\_SetTimer} are normally implemented by Tcl, but may be replaced with new versions to retarget the notifier (the \texttt{Tcl\_InitNotifier}, \texttt{Tcl\_AlertNotifier}, \texttt{Tcl\_FinalizeNotifier}, \texttt{Tcl\_Sleep}, \texttt{Tcl\_CreateFileHandler}, and \texttt{Tcl\_DeleteFileHandler} must also be replaced; see CREATING A NEW NOTIFIER below for details). The procedures \texttt{Tcl\_ServiceAll}, \texttt{Tcl\_ServiceEvent}, \texttt{Tcl\_GetServiceMode}, and \texttt{Tcl\_SetServiceMode} are provided to help connect Tcl's event loop to an external event loop such as Motif's.

NOTIFIER BASICS

The easiest way to understand how the notifier works is to consider what happens when \texttt{Tcl\_DoOneEvent} is called. \texttt{Tcl\_DoOneEvent} is passed a flags argument that indicates what sort of events it is OK to process and also whether or not to block if no events are ready. \texttt{Tcl\_DoOneEvent} does the following things:

1. Check the event queue to see if it contains any events that can be serviced. If so, service the first possible event, remove it from the queue, and return. It does this by calling \texttt{Tcl\_ServiceEvent} and passing in the flags argument.

2. Prepare to block for an event. To do this, \texttt{Tcl\_DoOneEvent} invokes a setup procedure in each event source. The event source will perform event−source specific initialization and possibly call \texttt{Tcl\_SetMaxBlockTime} to limit how long \texttt{Tcl\_WaitForEvent} will block if no new events occur.

3. Call \texttt{Tcl\_WaitForEvent}. This procedure is implemented differently on different platforms; it waits for an event to occur, based on the information provided by the event sources. It may cause the application to block if timePtr specifies an interval other than 0. \texttt{Tcl\_WaitForEvent} returns when something has happened, such as a file becoming readable or the interval given by timePtr expiring. If there are no events for \texttt{Tcl\_WaitForEvent} to wait for, so that it would block forever, then it returns immediately and \texttt{Tcl\_DoOneEvent} returns 0.

4. Call a check procedure in each event source. The check procedure determines whether any events of interest to this source occurred. If so, the events are added to the event queue.

5. Check the event queue to see if it contains any events that can be serviced. If so, service the first possible event, remove it from the queue, and return.

6. See if there are idle callbacks pending. If so, invoke all of them and return.

7. Either return 0 to indicate that no events were ready, or go back to step [2] if blocking was requested by the caller.

CREATING A NEW EVENT SOURCE

An event source consists of three procedures invoked by the notifier, plus additional C procedures that are invoked by higher−level code to arrange for event−driven callbacks. The three procedures called by the notifier consist of the setup and check procedures described above, plus an additional procedure that is
invoked when an event is removed from the event queue for servicing.

The procedure `Tcl_CreateEventSource` creates a new event source. Its arguments specify the setup procedure and check procedure for the event source. `SetupProc` should match the following prototype:

```c
typedef void Tcl_EventSetupProc(
    ClientData clientData,
    int flags);
```

The `clientData` argument will be the same as the `clientData` argument to `Tcl_CreateEventSource`; it is typically used to point to private information managed by the event source. The `flags` argument will be the same as the `flags` argument passed to `Tcl_DoOneEvent` except that it will never be 0 (`Tcl_DoOneEvent` replaces 0 with `TCL_ALL_EVENTS`). `Flags` indicates what kinds of events should be considered; if the bit corresponding to this event source isn't set, the event source should return immediately without doing anything. For example, the file event source checks for the `TCL_FILE_EVENTS` bit.

SetupProc's job is to make sure that the application wakes up when events of the desired type occur. This is typically done in a platform–dependent fashion. For example, under Unix an event source might call `Tcl_CreateFileHandler`; under Windows it might request notification with a Windows event. For timer–driven event sources such as timer events or any polled event, the event source can call `Tcl_SetMaxBlockTime` to force the application to wake up after a specified time even if no events have occurred. If no event source calls `Tcl_SetMaxBlockTime` then `Tcl_WaitForEvent` will wait as long as necessary for an event to occur; otherwise, it will only wait as long as the shortest interval passed to `Tcl_SetMaxBlockTime` by one of the event sources. If an event source knows that it already has events ready to report, it can request a zero maximum block time. For example, the setup procedure for the X event source looks to see if there are events already queued. If there are, it calls `Tcl_SetMaxBlockTime` with a 0 block time so that `Tcl_WaitForEvent` does not block if there is no new data on the X connection. The `timePtr` argument to `Tcl_WaitForEvent` points to a structure that describes a time interval in seconds and microseconds:

```c
typedef struct Tcl_Time {
    long sec;
    long usec;
} Tcl_Time;
```

The `usec` field should be less than 1000000.

Information provided to `Tcl_SetMaxBlockTime` is only used for the next call to `Tcl_WaitForEvent`; it is discarded after `Tcl_WaitForEvent` returns. The next time an event wait is done each of the event sources’ setup procedures will be called again, and they can specify new information for that event wait.

If the application uses an external event loop rather than `Tcl_DoOneEvent`, the event sources may need to call `Tcl_SetMaxBlockTime` at other times. For example, if a new event handler is registered that needs to poll for events, the event source may call `Tcl_SetMaxBlockTime` to set the block time to zero to force the external event loop to call Tcl. In this case, `Tcl_SetMaxBlockTime` invokes `Tcl_SetTimer` with the shortest interval seen since the last call to `Tcl_DoOneEvent` or `Tcl_ServiceAll`.

In addition to the generic procedure \texttt{Tcl\_SetMaxBlockTime}, other platform-specific procedures may also be available for \texttt{setupProc}, if there is additional information needed by \texttt{Tcl\_WaitForEvent} on that platform. For example, on Unix systems the \texttt{Tcl\_CreateFileHandler} interface can be used to wait for file events.

The second procedure provided by each event source is its check procedure, indicated by the \texttt{checkProc} argument to \texttt{Tcl\_CreateEventSource}. \texttt{CheckProc} must match the following prototype:

\begin{verbatim}
typedef void Tcl\_EventCheckProc(
    ClientData clientData,
    flags);
\end{verbatim}

The arguments to this procedure are the same as those for \texttt{setupProc}. \texttt{CheckProc} is invoked by \texttt{Tcl\_DoOneEvent} after it has waited for events. Presumably at least one event source is now prepared to queue an event. \texttt{Tcl\_DoOneEvent} calls each of the event sources in turn, so they all have a chance to queue any events that are ready. The check procedure does two things. First, it must see if any events have triggered. Different event sources do this in different ways.

If an event source's check procedure detects an interesting event, it must add the event to Tcl's event queue. To do this, the event source calls \texttt{Tcl\_QueueEvent}. The \texttt{evPtr} argument is a pointer to a dynamically allocated structure containing the event (see below for more information on memory management issues). Each event source can define its own event structure with whatever information is relevant to that event source. However, the first element of the structure must be a structure of type \texttt{Tcl\_Event}, and the address of this structure is used when communicating between the event source and the rest of the notifier. A \texttt{Tcl\_Event} has the following definition:

\begin{verbatim}
typedef struct {
    Tcl\_EventProc \*proc;
    struct Tcl\_Event \*nextPtr;
} Tcl\_Event;
\end{verbatim}

The event source must fill in the \texttt{proc} field of the event before calling \texttt{Tcl\_QueueEvent}. The \texttt{nextPtr} is used to link together the events in the queue and should not be modified by the event source.

An event may be added to the queue at any of three positions, depending on the \texttt{position} argument to \texttt{Tcl\_QueueEvent}:

\texttt{TCL\_QUEUE\_TAIL}
- Add the event at the back of the queue, so that all other pending events will be serviced first. This is almost always the right place for new events.

\texttt{TCL\_QUEUE\_HEAD}
- Add the event at the front of the queue, so that it will be serviced before all other queued events.

\texttt{TCL\_QUEUE\_MARK}
- Add the event at the front of the queue, unless there are other events at the front whose position is \texttt{TCL\_QUEUE\_MARK}; if so, add the new event just after all other \texttt{TCL\_QUEUE\_MARK} events. This value of \texttt{position} is used to insert an ordered sequence of events at the front of the queue, such as a series of Enter and Leave events synthesized during a grab or ungrab operation in Tk.
When it is time to handle an event from the queue (steps 1 and 4 above) `Tcl_ServiceEvent` will invoke the proc specified in the first queued `Tcl_Event` structure. Proc must match the following prototype:

```c
typedef int Tcl_EventProc(
    Tcl_Event * evPtr,
    int flags);
```

The first argument to proc is a pointer to the event, which will be the same as the first argument to the `Tcl_QueueEvent` call that added the event to the queue. The second argument to proc is the flags argument for the current call to `Tcl_ServiceEvent`; this is used by the event source to return immediately if its events are not relevant.

It is up to proc to handle the event, typically by invoking one or more Tcl commands or C−level callbacks. Once the event source has finished handling the event it returns 1 to indicate that the event can be removed from the queue. If for some reason the event source decides that the event cannot be handled at this time, it may return 0 to indicate that the event should be deferred for processing later; in this case `Tcl_ServiceEvent` will go on to the next event in the queue and attempt to service it. There are several reasons why an event source might defer an event. One possibility is that events of this type are excluded by the flags argument. For example, the file event source will always return 0 if the `TCL_FILE_EVENTS` bit isn't set in flags. Another example of deferring events happens in Tk if `Tk_RestrictEvents` has been invoked to defer certain kinds of window events.

When proc returns 1, `Tcl_ServiceEvent` will remove the event from the event queue and free its storage. Note that the storage for an event must be allocated by the event source (using `Tcl_Alloc` or the Tcl macro `ckalloc`) before calling `Tcl_QueueEvent`, but it will be freed by `Tcl_ServiceEvent`, not by the event source.

Threaded applications work in a similar manner, except that there is a separate event queue for each thread containing a Tcl interpreter. Calling `Tcl_QueueEvent` in a multithreaded application adds an event to the current thread's queue. To add an event to another thread's queue, use `Tcl_ThreadQueueEvent`. `Tcl_ThreadQueueEvent` accepts as an argument a Tcl_ThreadID argument, which uniquely identifies a thread in a Tcl application. To obtain the Tcl_ThreadID for the current thread, use the `Tcl_GetCurrentThread` procedure. (A thread would then need to pass this identifier to other threads for those threads to be able to add events to its queue.) After adding an event to another thread's queue, you then typically need to call `Tcl_ThreadAlert` to "wake up" that thread's notifier to alert it to the new event.

`Tcl_DeleteEvents` can be used to explicitly remove one or more events from the event queue. `Tcl_DeleteEvents` calls proc for each event in the queue, deleting those for with the procedure returns 1. Events for which the procedure returns 0 are left in the queue. Proc should match the following prototype:

```c
typedef int Tcl_EventDeleteProc(
    Tcl_Event * evPtr,
    ClientData clientData);
```

The clientData argument will be the same as the clientData argument to `Tcl_DeleteEvents`; it is typically used to point to private information managed by the event source. The evPtr will point to the next event in the queue.
**Tcl_DeleteEventSource** deletes an event source. The *setupProc*, *checkProc*, and *clientData* arguments must exactly match those provided to the **Tcl_CreateEventSource** for the event source to be deleted. If no such source exists, **Tcl_DeleteEventSource** has no effect.

### CREATING A NEW NOTIFIER

The notifier consists of all the procedures described in this manual entry, plus **Tcl_DoOneEvent** and **Tcl_Sleep**, which are available on all platforms, and **Tcl_CreateFileHandler** and **Tcl_DeleteFileHandler**, which are Unix-specific. Most of these procedures are generic, in that they are the same for all notifiers. However, eight of the procedures are notifier-dependent: **Tcl_InitNotifier**, **Tcl_AlertNotifier**, **Tcl_FinalizeNotifier**, **Tcl_SetTimer**, **Tcl_DoOneEvent**, **Tcl_Sleep**, **Tcl_CreateFileHandler** and **Tcl_DeleteFileHandler**. To support a new platform or to integrate Tcl with an application-specific event loop, you must write new versions of these procedures.

**Tcl_InitNotifier** initializes the notifier state and returns a handle to the notifier state. Tcl calls this procedure when initializing a Tcl interpreter. Similarly, **Tcl_FinalizeNotifier** shuts down the notifier, and is called by **Tcl_Finalize** when shutting down a Tcl interpreter.

**Tcl_WaitForEvent** is the lowest-level procedure in the notifier; it is responsible for waiting for an "interesting" event to occur or for a given time to elapse. Before **Tcl_WaitForEvent** is invoked, each of the event sources' setup procedure will have been invoked. The *timePtr* argument to **Tcl_WaitForEvent** gives the maximum time to block for an event, based on calls to **Tcl_SetMaxBlockTime** made by setup procedures and on other information (such as the **TCL_DONT_WAIT** bit in flags).

Ideally, **Tcl_WaitForEvent** should only wait for an event to occur; it should not actually process the event in any way. Later on, the event sources will process the raw events and create Tcl_Events on the event queue in their *checkProc* procedures. However, on some platforms (such as Windows) this isn't possible; events may be processed in **Tcl_WaitForEvent**, including queuing Tcl_Events and more (for example, callbacks for native widgets may be invoked). The return value from **Tcl_WaitForEvent** must be either 0, 1, or −1. On platforms such as Windows where events get processed in **Tcl_WaitForEvent**, a return value of 1 means that there may be more events still pending that haven't been processed. This is a sign to the caller that it must call **Tcl_WaitForEvent** again if it wants all pending events to be processed. A 0 return value means that calling **Tcl_WaitForEvent** again will not have any effect: either this is a platform where **Tcl_WaitForEvent** only waits without doing any event processing, or **Tcl_WaitForEvent** knows for sure that there are no additional events to process (e.g. it returned because the time elapsed). Finally, a return value of −1 means that the event loop is no longer operational and the application should probably unwind and terminate. Under Windows this happens when a WM_QUIT message is received; under Unix it happens when **Tcl_WaitForEvent** would have waited forever because there were no active event sources and the timeout was infinite.

**Tcl_AlertNotifier** is used in multithreaded applications to allow any thread to "wake up" the notifier to alert it to new events on its queue. **Tcl_AlertNotifier** requires as an argument the notifier handle returned by **Tcl_InitNotifier**.

If the notifier will be used with an external event loop, then it must also support the **Tcl_SetTimer** interface. **Tcl_SetTimer** is invoked by **Tcl_SetMaxBlockTime** whenever the maximum blocking time has been
reduced. **Tcl_SetTimer** should arrange for the external event loop to invoke **Tcl_ServiceAll** after the specified interval even if no events have occurred. This interface is needed because **Tcl_WaitForEvent** isn't invoked when there is an external event loop. If the notifier will only be used from **Tcl_DoOneEvent**, then **Tcl_SetTimer** need not do anything.

On Unix systems, the file event source also needs support from the notifier. The file event source consists of the **Tcl_CreateFileHandler** and **Tcl_DeleteFileHandler** procedures, which are described in the **Tcl_CreateFileHandler** manual page.

The **Tcl_Sleep** and **Tcl_DoOneEvent** interfaces are described in their respective manual pages.

The easiest way to create a new notifier is to look at the code for an existing notifier, such as the files unix/tclUnixNotify.c or win/tclWinNotify.c in the Tcl source distribution.

### EXTERNAL EVENT LOOPS

The notifier interfaces are designed so that Tcl can be embedded into applications that have their own private event loops. In this case, the application does not call **Tcl_DoOneEvent** except in the case of recursive event loops such as calls to the Tcl commands **update** or **vwait**. Most of the time is spent in the external event loop of the application. In this case the notifier must arrange for the external event loop to call back into Tcl when something happens on the various Tcl event sources. These callbacks should arrange for appropriate Tcl events to be placed on the Tcl event queue.

Because the external event loop is not calling **Tcl_DoOneEvent** on a regular basis, it is up to the notifier to arrange for **Tcl_ServiceEvent** to be called whenever events are pending on the Tcl event queue. The easiest way to do this is to invoke **Tcl_ServiceAll** at the end of each callback from the external event loop. This will ensure that all of the event sources are polled, any queued events are serviced, and any pending idle handlers are processed before returning control to the application. In addition, event sources that need to poll for events can call **Tcl_SetMaxBlockTime** to force the external event loop to call Tcl even if no events are available on the system event queue.

As a side effect of processing events detected in the main external event loop, Tcl may invoke **Tcl_DoOneEvent** to start a recursive event loop in commands like **vwait Tcl_DoOneEvent** will invoke the external event loop, which will result in callbacks as described in the preceding paragraph, which will result in calls to **Tcl_ServiceAll**. However, in these cases it is undesirable to service events in **Tcl_ServiceAll**. Servicing events there is unnecessary because control will immediately return to the external event loop and hence to **Tcl_DoOneEvent**, which can service the events itself. Furthermore, **Tcl_DoOneEvent** is supposed to service only a single event, whereas **Tcl_ServiceAll** normally services all pending events. To handle this situation, **Tcl_DoOneEvent** sets a flag for **Tcl_ServiceAll** that causes it to return without servicing any events. This flag is called the *service mode*; **Tcl_DoOneEvent** restores it to its previous value before it returns.

In some cases, however, it may be necessary for **Tcl_ServiceAll** to service events even when it has been invoked from **Tcl_DoOneEvent**. This happens when there is yet another recursive event loop invoked via an event handler called by **Tcl_DoOneEvent** (such as one that is part of a native widget). In this case,
Tcl_DoOneEvent may not have a chance to service events so Tcl_ServiceAll must service them all. Any recursive event loop that calls an external event loop rather than Tcl_DoOneEvent must reset the service mode so that all events get processed in Tcl_ServiceAll. This is done by invoking the Tcl_SetServiceMode procedure. If Tcl_SetServiceMode is passed TCL_SERVICE_NONE, then calls to Tcl_ServiceAll will return immediately without processing any events. If Tcl_SetServiceMode is passed TCL_SERVICE_ALL, then calls to Tcl_ServiceAll will behave normally. Tcl_SetServiceMode returns the previous value of the service mode, which should be restored when the recursive loop exits. Tcl_GetServiceMode returns the current value of the service mode.

SEE ALSO

Tcl_CreateFileHandler, Tcl_DeleteFileHandler, Tcl_Sleep, Tcl_DoOneEvent, Thread

KEYWORDS

event, notifier, event queue, event sources, file events, timer, idle, service mode, threads
NAME

Tcl_StandardChannels – How the Tcl library deals with the standard channels

DESCRIPTION

APIs

INITIALIZATION OF TCL STANDARD CHANNELS

1)

2)

(a)

(b)

RE-INITIALIZATION OF TCL STANDARD CHANNELS

tclsh

wish

SEE ALSO

KEYWORDS

NAME

Tcl_StandardChannels – How the Tcl library deals with the standard channels

DESCRIPTION

This page explains the initialization and use of standard channels in the Tcl library.

The term standard channels comes out of the Unix world and refers to the three channels automatically opened by the OS for each new application. They are stdin, stdout and stderr. The first is the standard input an application can read from, the other two refer to writable channels, one for regular output and the other for error messages.

Tcl generalizes this concept in a cross-platform way and exposes standard channels to the script level.

APIs

The public API procedures dealing directly with standard channels are Tcl_GetStdChannel and Tcl_SetStdChannel. Additional public APIs to consider are Tcl_RegisterChannel, Tcl_CreateChannel and Tcl_GetChannel.
INITIALIZATION OF TCL STANDARD CHANNELS

Standard channels are initialized by the Tcl library in three cases: when explicitly requested, when implicitly required before returning channel information, or when implicitly required during registration of a new channel.

These cases differ in how they handle unavailable platform-specific standard channels. (A channel is not `available` if it could not be successfully opened; for example, in a Tcl application run as a Windows NT service.)

1) A single standard channel is initialized when it is explicitly specified in a call to `Tcl_SetStdChannel`. The state of the other standard channels are unaffected.

   Missing platform-specific standard channels do not matter here. This approach is not available at the script level.

2) All uninitialized standard channels are initialized to platform-specific default values:
   
   (a) when open channels are listed with `Tcl_GetChannelNames` (or the file channels script command), or
   
   (b) when information about any standard channel is requested with a call to `Tcl_GetStdChannel`, or with a call to `Tcl_GetChannel` which specifies one of the standard names (`stdin`, `stdout` and `stderr`).

   In case of missing platform-specific standard channels, the Tcl standard channels are considered as initialized and then immediately closed. This means that the first three Tcl channels then opened by the application are designated as the Tcl standard channels.

3) All uninitialized standard channels are initialized to platform-specific default values when a user-requested channel is registered with `Tcl_RegisterChannel`.

   In case of unavailable platform-specific standard channels the channel whose creation caused the initialization of the Tcl standard channels is made a normal channel. The next three Tcl channels opened by the application are designated as the Tcl standard channels. In other words, of the first four Tcl channels opened by the application the second to fourth are designated as the Tcl standard channels.

RE-INITIALIZATION OF TCL STANDARD CHANNELS

Once a Tcl standard channel is initialized through one of the methods above, closing this Tcl standard channel will cause the next call to `Tcl_CreateChannel` to make the new channel the new standard channel, too. If more than one Tcl standard channel was closed `Tcl_CreateChannel` will fill the empty slots in the order `stdin`, `stdout` and `stderr`. 
**Tcl_CreateChannel** will not try to reinitialize an empty slot if that slot was not initialized before. It is this behavior which enables an application to employ method 1 of initialization, i.e. to create and designate their own Tcl standard channels.

**tclsh**

The Tcl shell (or rather **Tcl_Main**) uses method 2 to initialize the standard channels.

**wish**

The windowing shell (or rather **Tk_MainEx**) uses method 1 to initialize the standard channels (See **Tk_InitConsoleChannels**) on non–Unix platforms. On Unix platforms, **Tk_MainEx** implicitly uses method 2 to initialize the standard channels.

**SEE ALSO**

**Tcl_CreateChannel, Tcl_RegisterChannel, Tcl_GetChannel, Tcl_GetStdChannel, Tcl_SetStdChannel, Tk_InitConsoleChannels**, **tclsh, wish, Tcl_Main, Tk_MainEx**

**KEYWORDS**

standard channels
TCL_MEM_DEBUG

NAME

TCL_MEM_DEBUG – Compile–time flag to enable Tcl memory debugging.

DESCRIPTION

When Tcl is compiled with TCL_MEM_DEBUG defined, a powerful set of memory debugging aids are included in the compiled binary. This includes C and Tcl functions which can aid with debugging memory leaks, memory allocation overruns, and other memory related errors.

ENABLING MEMORY DEBUGGING

To enable memory debugging, Tcl should be recompiled from scratch with TCL_MEM_DEBUG defined. This will also compile in a non–stub version of Tcl_InitMemory to add the memory command to Tcl.

TCL_MEM_DEBUG must be either left defined for all modules or undefined for all modules that are going to be linked together. If they are not, link errors will occur, with either TclDbCkfree and Tcl_DbCkalloc or Tcl_Ckalloc and Tcl_Ckfree being undefined.

Once memory debugging support has been compiled into Tcl, the C functions Tcl.ValidateAllMemory and Tcl.DumpActiveMemory, and the Tcl memory command can be used to validate and examine memory usage.

GUARD ZONES

When memory debugging is enabled, whenever a call to ckalloc is made, slightly more memory than requested is allocated so the memory debugging code can keep track of the allocated memory, and eight–byte “guard zones” are placed in front of and behind the space that will be returned to the caller. (The sizes of the guard zones are defined by the C #define LOW_GUARD_SIZE and #define HIGH_GUARD_SIZE in the file generic/tclCkalloc.c — it can be extended if you suspect large overwrite problems, at some cost in performance.) A known pattern is written into the guard zones and, on a call to ckfree, the guard zones of the space being freed are checked to see if either zone has been modified in any way. If one has been, the guard bytes and their new contents are identified, and a “low guard failed” or “high guard failed” message is issued. The “guard failed” message includes the address of the memory packet and the file name and line number of the code that called ckfree. This allows you to detect the common sorts of one–off problems, where not enough space was allocated to contain the data written, for example.

DEBUGGING DIFFICULT MEMORY CORRUPTION PROBLEMS

Normally, Tcl compiled with memory debugging enabled will make it easy to isolate a corruption problem. Turning on memory validation with the memory command can help isolate difficult problems. If you suspect (or know) that corruption is occurring before the Tcl interpreter comes up far enough for you to issue
commands, you can set MEM_VALIDATE define, recompile tclCkalloc.c and rebuild Tcl. This will enable memory validation from the first call to `ckalloc`, again, at a large performance impact.

If you are desperate and validating memory on every call to `ckalloc` and `ckfree` isn't enough, you can explicitly call `Tcl_ValidateAllMemory` directly at any point. It takes a `char *` and an `int` which are normally the filename and line number of the caller, but they can actually be anything you want. Remember to remove the calls after you find the problem.

**SEE ALSO**

`ckalloc`, `memory`, `Tcl_ValidateAllMemory`, `Tcl_DumpActiveMemory`

**KEYWORDS**

`memory`, `debug`

---

Copyright ' 1992−1999 Karl Lehenbauer and Mark Diekhans.
Copyright ' 2000 by Scriptics Corporation.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
Access

NAME

Tcl_Access, Tcl_Stat – check file permissions and other attributes

SYNOPSIS

#include <tcl.h>

int Tcl_Access(path, mode)

int Tcl_Stat(path, statPtr)

ARGUMENTS

char *path (in)

Native name of the file to check the attributes of.

int mode (in)

Mask consisting of one or more of R_OK, W_OK, X_OK and F_OK. R_OK, W_OK and X_OK request checking whether the file exists and has read, write and execute permissions, respectively. F_OK just requests checking for the existence of the file.

struct stat *statPtr (out)

The structure that contains the result.

DESCRIPTION

As of Tcl 8.4, the object–based APIs Tcl_FSAccess and Tcl_FSStat should be used in preference to Tcl_Access and Tcl_Stat, wherever possible.

There are two reasons for calling Tcl_Access and Tcl_Stat rather than calling system level functions access and stat directly. First, the Windows implementation of both functions fixes some bugs in the system level calls. Second, both Tcl_Access and Tcl_Stat (as well as Tcl_OpenFileChannelProc) hook into a linked list of functions. This allows the possibility to reroute file access to alternative media or access methods.

Tcl_Access checks whether the process would be allowed to read, write or test for existence of the file (or other file system object) whose name is pathname. If pathname is a symbolic link on Unix, then permissions of the file referred by this symbolic link are tested.

On success (all requested permissions granted), zero is returned. On error (at least one bit in mode asked for a permission that is denied, or some other error occurred), −1 is returned.

Tcl_Stat fills the stat structure statPtr with information about the specified file. You do not need any access rights to the file to get this information but you need search rights to all directories named in the path leading...
to the file. The stat structure includes info regarding device, inode (always 0 on Windows), privilege mode, nlink (always 1 on Windows), user id (always 0 on Windows), group id (always 0 on Windows), rdev (same as device on Windows), size, last access time, last modification time, and creation time.

If *path* exists, **Tcl_Stat** returns 0 and the stat structure is filled with data. Otherwise, −1 is returned, and no stat info is given.

**KEYWORDS**

*stat, access*
AddErrInfo

NAME

Tcl_AddObjErrorInfo, Tcl_AddErrorInfo, Tcl_SetObjErrorCode, Tcl_SetErrorCode, Tcl_SetErrorCodeVA, Tcl_PosixError, Tcl_LogCommandInfo – record information about errors

SYNOPSIS

#include <tcl.h>
Tcl_AddObjErrorInfo(interp, message, length)
Tcl_AddErrorInfo(interp, message)
Tcl_SetObjErrorCode(interp, errorObjPtr)
Tcl_SetErrorCode(interp, element, element, ... (char *) NULL)
Tcl_SetErrorCodeVA(interp, argList)
CONST char *
Tcl_PosixError(interp)
void
Tcl_LogCommandInfo(interp, script, command, commandLength)

ARGUMENTS

Tcl_Interp *interp (in)
    Interpreter in which to record information.
char *message (in)
    For Tcl_AddObjErrorInfo, this points to the first byte of an array of bytes containing a string to record in the errorInfo variable. This byte array may contain embedded null bytes unless length is negative. For Tcl_AddErrorInfo, this is a conventional C string to record in the errorInfo variable.
int length (in)
    The number of bytes to copy from message when setting the errorInfo variable. If negative, all bytes up to the first null byte are used.
Tcl_Obj *errorObjPtr (in)
    This variable errorCode will be set to this value.
char *element (in)
    String to record as one element of errorCode variable. Last element argument must be NULL.
va_list argList (in)
    An argument list which must have been initialized using TCL_VARARGS_START, and cleared using va_end.
CONST char *script (in)
    Pointer to first character in script containing command (must be <= command)
CONST char *command (in)
    Pointer to first character in command that generated the error
int commandLength (in)
    Number of bytes in command; −1 means use all bytes up to first null byte
DESCRIPTION

These procedures are used to manipulate two Tcl global variables that hold information about errors. The variable errorInfo holds a stack trace of the operations that were in progress when an error occurred, and is intended to be human-readable. The variable errorCode holds a list of items that are intended to be machine-readable. The first item in errorCode identifies the class of error that occurred (e.g. POSIX means an error occurred in a POSIX system call) and additional elements in errorCode hold additional pieces of information that depend on the class. See the Tcl overview manual entry for details on the various formats for errorCode.

The errorInfo variable is gradually built up as an error unwinds through the nested operations. Each time an error code is returned to Tcl_EvalObjEx (or Tcl_Eval, which calls Tcl_EvalObjEx) it calls the procedure Tcl_AddObjErrorInfo to add additional text to errorInfo describing the command that was being executed when the error occurred. By the time the error has been passed all the way back to the application, it will contain a complete trace of the activity in progress when the error occurred.

It is sometimes useful to add additional information to errorInfo beyond what can be supplied automatically by Tcl_EvalObjEx. Tcl_AddObjErrorInfo may be used for this purpose: its message and length arguments describe an additional string to be appended to errorInfo. For example, the source command calls Tcl_AddObjErrorInfo to record the name of the file being processed and the line number on which the error occurred; for Tcl procedures, the procedure name and line number within the procedure are recorded, and so on. The best time to call Tcl_AddObjErrorInfo is just after Tcl_EvalObjEx has returned TCL_ERROR. In calling Tcl_AddObjErrorInfo, you may find it useful to use the errorLine field of the interpreter (see the Tcl_Interp manual entry for details).

Tcl_AddErrorInfo resembles Tcl_AddObjErrorInfo but differs in initializing errorInfo from the string value of the interpreter's result if the error is just starting to be logged. It does not use the result as a Tcl object so any embedded null characters in the result will cause information to be lost. It also takes a conventional C string in message instead of Tcl_AddObjErrorInfo's counted string.

The procedure Tcl_SetObjErrorCode is used to set the errorCode variable. errorObjPtr contains a list object built up by the caller. errorCode is set to this value. Tcl_SetObjErrorCode is typically invoked just before returning an error in an object command. If an error is returned without calling Tcl_SetObjErrorCode or Tcl_SetErrorCode the Tcl interpreter automatically sets errorCode to NONE.

The procedure Tcl_SetErrorCode is also used to set the errorCode variable. However, it takes one or more strings to record instead of an object. Otherwise, it is similar to Tcl_SetObjErrorCode in behavior.

Tcl_SetErrorCodeVA is the same as Tcl_SetErrorCode except that instead of taking a variable number of arguments it takes an argument list.

Tcl_PosixError sets the errorCode variable after an error in a POSIX kernel call. It reads the value of the errno C variable and calls Tcl_SetErrorCode to set errorCode in the POSIX format. The caller must previously have called Tcl_SetErrno to set errno; this is necessary on some platforms (e.g. Windows) where Tcl is linked into an application as a shared library, or when the error occurs in a dynamically loaded
extension. See the manual entry for `Tcl_SetErrno` for more information.

`Tcl_PosixError` returns a human-readable diagnostic message for the error (this is the same value that will appear as the third element in `errorCode`). It may be convenient to include this string as part of the error message returned to the application in the interpreter's result.

`Tcl_LogCommandInfo` is invoked after an error occurs in an interpreter. It adds information about the command that was being executed when the error occurred to the `errorInfo` variable, and the line number stored internally in the interpreter is set. On the first call to `Tcl_LogCommandInfo` or `Tcl_AddObjErrorInfo` since an error occurred, the old information in `errorInfo` is deleted.

It is important to call the procedures described here rather than setting `errorInfo` or `errorCode` directly with `Tcl_ObjSetVar2`. The reason for this is that the Tcl interpreter keeps information about whether these procedures have been called. For example, the first time `Tcl_AddObjErrorInfo` is called for an error, it clears the existing value of `errorInfo` and adds the error message in the interpreter's result to the variable before appending `message`; in subsequent calls, it just appends the new `message`. When `Tcl_SetErrorCode` is called, it sets a flag indicating that `errorCode` has been set; this allows the Tcl interpreter to set `errorCode` to `NONE` if it receives an error return when `Tcl_SetErrorCode` hasn't been called.

If the procedure `Tcl_ResetResult` is called, it clears all of the state associated with `errorInfo` and `errorCode` (but it doesn't actually modify the variables). If an error had occurred, this will clear the error state to make it appear as if no error had occurred after all.

SEE ALSO

`Tcl-DecrRefCount`, `Tcl_IncrRefCount`, `Tcl_Interp`, `Tcl_ResetResult`, `Tcl_SetErrno`

KEYWORDS

`error`, `object`, `object result`, `stack`, `trace`, `variable`
Alloc

NAME

Tcl_Alloc, Tcl_Free, Tcl_Realloc, Tcl_AttemptAlloc, Tcl_AttemptRealloc, ckalloc, ckfree, ckrealloc, attemptckalloc, attemptckrealloc – allocate or free heap memory

SYNOPSIS

#include <tcl.h>
char *
Tcl_Alloc(size)
void
Tcl_Free(ptr)
char *
Tcl_Realloc(ptr, size)
char *
Tcl_AttemptAlloc(size)
char *
Tcl_AttemptRealloc(ptr, size)
char *
ckalloc(size)
void
ckfree(ptr)
char *
ckrealloc(ptr, size)
char *
attemptckalloc(size)
char *
attemptckrealloc(ptr, size)

ARGUMENTS

int size (in)
  Size in bytes of the memory block to allocate.
char *ptr (in)
  Pointer to memory block to free or realloc.

DESCRIPTION

These procedures provide a platform and compiler independent interface for memory allocation. Programs that need to transfer ownership of memory blocks between Tcl and other modules should use these routines rather than the native malloc() and free() routines provided by the C run–time library.
Tcl_Alloc returns a pointer to a block of at least size bytes suitably aligned for any use.

Tcl_Free makes the space referred to by ptr available for further allocation.

Tcl_Realloc changes the size of the block pointed to by ptr to size bytes and returns a pointer to the new block. The contents will be unchanged up to the lesser of the new and old sizes. The returned location may be different from ptr.

Tcl_AttemptAlloc and Tcl_AttemptRealloc are identical in function to Tcl_Alloc and Tcl_Realloc, except that Tcl_AttemptAlloc and Tcl_AttemptRealloc will not cause the Tcl interpreter to panic if the memory allocation fails. If the allocation fails, these functions will return NULL. Note that on some platforms, attempting to allocate a block of memory will also cause these functions to return NULL.

The procedures ckalloc, ckfree, ckrealloc, attemptckalloc, and attemptckrealloc are implemented as macros. Normally, they are synonyms for the corresponding procedures documented on this page. When Tcl and all modules calling Tcl are compiled with TCL_MEM_DEBUG defined, however, these macros are redefined to be special debugging versions of these procedures. To support Tcl's memory debugging within a module, use the macros rather than direct calls to Tcl_Alloc, etc.

KEYWORDS

alloc, allocation, free, malloc, memory, realloc, TCL_MEM_DEBUG

Copyright © 1995-1996 Sun Microsystems, Inc.
Copyright © 1995-1997 Roger E. Critchlow Jr.
Tcl_AllowExceptions − allow all exceptions in next script evaluation

SYNOPSIS

#include <tcl.h>
Tcl_AllowExceptions(interp)

ARGUMENTS

Tcl_Interp *interp (in)
Interpreter in which script will be evaluated.

DESCRIPTION

If a script is evaluated at top−level (i.e. no other scripts are pending evaluation when the script is invoked), and if the script terminates with a completion code other than TCL_OK, TCL_ERROR or TCL_RETURN, then Tcl normally converts this into a TCL_ERROR return with an appropriate message. The particular script evaluation procedures of Tcl that act in the manner are Tcl_EvalObjEx, Tcl_EvalObjv, Tcl_Eval, Tcl_EvalEx, Tcl_GlobalEval, Tcl_GlobalEvalObj, Tcl_VarEval and Tcl_VarEvalVA.

However, if Tcl_AllowExceptions is invoked immediately before calling one of those procedures, then arbitrary completion codes are permitted from the script, and they are returned without modification. This is useful in cases where the caller can deal with exceptions such as TCL_BREAK or TCL_CONTINUE in a meaningful way.

KEYWORDS

continue, break, exception, interpreter

Copyright © 1989−1993 The Regents of the University of California.
Copyright © 1994−1996 Sun Microsystems, Inc.
Copyright © 1995−1997 Roger E. Critchlow Jr.
ApplInit

NAME

Tcl_AppInit – perform application–specific initialization

SYNOPSIS

#include <tcl.h>

int Tcl_AppInit(interp)

ARGUMENTS

interp (in)
Interpreter for the application.

DESCRIPTION

Tcl_AppInit is a “hook” procedure that is invoked by the main programs for Tcl applications such as tclsh and wish. Its purpose is to allow new Tcl applications to be created without modifying the main programs provided as part of Tcl and Tk. To create a new application you write a new version of Tcl_AppInit to replace the default version provided by Tcl, then link your new Tcl_AppInit with the Tcl library.

Tcl_AppInit is invoked after by Tcl_Main and Tk_Main after their own initialization and before entering the main loop to process commands. Here are some examples of things that Tcl_AppInit might do:

[1] Call initialization procedures for various packages used by the application. Each initialization procedure adds new commands to interp for its package and performs other package–specific initialization.

[2] Process command–line arguments, which can be accessed from the Tcl variables argv and argv0 in interp.

[3] Invoke a startup script to initialize the application.

Tcl_AppInit returns TCL_OK or TCL_ERROR. If it returns TCL_ERROR then it must leave an error message in for the interpreter's result; otherwise the result is ignored.

In addition to Tcl_AppInit, your application should also contain a procedure main that calls Tcl_Main as follows:

Tcl_Main(argc, argv, Tcl_AppInit);
The third argument to `Tcl_Main` gives the address of the application–specific initialization procedure to invoke. This means that you don't have to use the name `Tcl_AppInit` for the procedure, but in practice the name is nearly always `Tcl_AppInit` (in versions before Tcl 7.4 the name `Tcl_AppInit` was implicit; there was no way to specify the procedure explicitly). The best way to get started is to make a copy of the file `tclAppInit.c` from the Tcl library or source directory. It already contains a `main` procedure and a template for `Tcl_AppInit` that you can modify for your application.

**KEYWORDS**

application, argument, command, initialization, interpreter

*Copyright ' 1993 The Regents of the University of California.*

*Copyright ' 1994-1996 Sun Microsystems, Inc.*

*Copyright ' 1995-1997 Roger E. Critchlow Jr.*
Async

NAME

Tcl_AsyncCreate, Tcl_AsyncMark, Tcl_AsyncInvoke, Tcl_AsyncDelete, Tcl_AsyncReady – handle asynchronous events

SYNOPSIS

#include <tcl.h>
Tcl_AsyncHandler
Tcl_AsyncCreate(proc, clientData)
Tcl_AsyncMark(async)
int
Tcl_AsyncInvoke(interp, code)
Tcl_AsyncDelete(async)
int
Tcl_AsyncReady()

ARGUMENTS

Tcl_AsyncProc *proc (in)
    Procedure to invoke to handle an asynchronous event.
ClientData clientData (in)
    One–word value to pass to proc.
Tcl_AsyncHandler async (in)
    Token for asynchronous event handler.
Tcl_Interp *interp (in)
    Tcl interpreter in which command was being evaluated when handler was invoked, or NULL if handler was invoked when there was no interpreter active.
int code (in)
    Completion code from command that just completed in interp, or 0 if interp is NULL.

DESCRIPTION

These procedures provide a safe mechanism for dealing with asynchronous events such as signals. If an event such as a signal occurs while a Tcl script is being evaluated then it isn't safe to take any substantive action to process the event. For example, it isn't safe to evaluate a Tcl script since the interpreter may already be in the middle of evaluating a script; it may not even be safe to allocate memory, since a memory allocation could have been in progress when the event occurred. The only safe approach is to set a flag indicating that the event occurred, then handle the event later when the world has returned to a clean state, such as after the current Tcl command completes.
Tcl AsyncCreate, Tcl AsyncDelete, and Tcl AsyncReady are thread sensitive. They access and/or set a thread-specific data structure in the event of an --enable-thread built core. The token created by TclAsyncCreate contains the needed thread information it was called from so that calling TclAsyncMark(token) will only yield the origin thread into the AsyncProc.

TclAsyncCreate creates an asynchronous handler and returns a token for it. The asynchronous handler must be created before any occurrences of the asynchronous event that it is intended to handle (it is not safe to create a handler at the time of an event). When an asynchronous event occurs the code that detects the event (such as a signal handler) should call TclAsyncMark with the token for the handler. TclAsyncMark will mark the handler as ready to execute, but it will not invoke the handler immediately. Tcl will call the proc associated with the handler later, when the world is in a safe state, and proc can then carry out the actions associated with the asynchronous event. Proc should have arguments and result that match the type TclAsyncProc:

```c
typedef int Tcl_AsyncProc(
    ClientData clientData,
    Tcl_Interp *interp,
    int code
);
```

The clientData will be the same as the clientData argument passed to TclAsyncCreate when the handler was created. If proc is invoked just after a command has completed execution in an interpreter, then interp will identify the interpreter in which the command was evaluated and code will be the completion code returned by that command. The command's result will be present in the interpreter's result. When proc returns, whatever it leaves in the interpreter's result will be returned as the result of the command and the integer value returned by proc will be used as the new completion code for the command.

It is also possible for proc to be invoked when no interpreter is active. This can happen, for example, if an asynchronous event occurs while the application is waiting for interactive input or an X event. In this case interp will be NULL and code will be 0, and the return value from proc will be ignored.

The procedure TclAsyncInvoke is called to invoke all of the handlers that are ready. The procedure TclAsyncReady will return non-zero whenever any asynchronous handlers are ready; it can be checked to avoid calls to TclAsyncInvoke when there are no ready handlers. Tcl calls TclAsyncReady after each command is evaluated and calls TclAsyncInvoke if needed. Applications may also call TclAsyncInvoke at interesting times for that application. For example, Tcl's event handler calls TclAsyncReady after each event and calls TclAsyncInvoke if needed. The interp and code arguments to TclAsyncInvoke have the same meaning as for proc: they identify the active interpreter, if any, and the completion code from the command that just completed.

TclAsyncDelete removes an asynchronous handler so that its proc will never be invoked again. A handler can be deleted even when ready, and it will still not be invoked.

If multiple handlers become active at the same time, the handlers are invoked in the order they were created (oldest handler first). The code and the interpreter's result for later handlers reflect the values returned by earlier handlers, so that the most recently created handler has last say about the interpreter's result and completion code. If new handlers become ready while handlers are executing, TclAsyncInvoke will invoke
them all; at each point it invokes the highest-priority (oldest) ready handler, repeating this over and over until there are no longer any ready handlers.

**WARNING**

It is almost always a bad idea for an asynchronous event handler to modify the interpreter's result or return a code different from its `code` argument. This sort of behavior can disrupt the execution of scripts in subtle ways and result in bugs that are extremely difficult to track down. If an asynchronous event handler needs to evaluate Tcl scripts then it should first save the interpreter's result plus the values of the variables `errorInfo` and `errorCode` (this can be done, for example, by storing them in dynamic strings). When the asynchronous handler is finished it should restore the interpreter's result, `errorInfo`, and `errorCode`, and return the `code` argument.

**KEYWORDS**

asynchronous event, handler, signal

---

*Copyright ' 1989-1993 The Regents of the University of California.*

*Copyright ' 1994-1996 Sun Microsystems, Inc.*

*Copyright ' 1995-1997 Roger E. Critchlow Jr.*
Tcl_BackgroundError − report Tcl error that occurred in background processing

SYNOPSIS

#include <tcl.h>
Tcl_BackgroundError(interp)

ARGUMENTS

Tcl_Interp *interp (in)
    Interpreter in which the error occurred.

DESCRIPTION

This procedure is typically invoked when a Tcl error occurs during "background processing" such as executing an event handler. When such an error occurs, the error condition is reported to Tcl or to a widget or some other C code, and there is not usually any obvious way for that code to report the error to the user. In these cases the code calls Tcl_BackgroundError with an interp argument identifying the interpreter in which the error occurred. At the time Tcl_BackgroundError is invoked, the interpreter's result is expected to contain an error message. Tcl_BackgroundError will invoke the bgerror Tcl command to report the error in an application−specific fashion. If no bgerror command exists, or if it returns with an error condition, then Tcl_BackgroundError reports the error itself by printing a message on the standard error file.

Tcl_BackgroundError does not invoke bgerror immediately because this could potentially interfere with scripts that are in process at the time the error occurred. Instead, it invokes bgerror later as an idle callback. Tcl_BackgroundError saves the values of the errorInfo and errorCode variables and restores these values just before invoking bgerror.

It is possible for many background errors to accumulate before bgerror is invoked. When this happens, each of the errors is processed in order. However, if bgerror returns a break exception, then all remaining error reports for the interpreter are skipped.

KEYWORDS

background, bgerror, error
Backslash

NAME

Tcl_Backslash – parse a backslash sequence

SYNOPSIS

#include <tcl.h>
char
Tcl_Backslash(src, countPtr)

ARGUMENTS

char *src (in)
     Pointer to a string starting with a backslash.
int *countPtr (out)
     If countPtr isn't NULL, *countPtr gets filled in with number of characters in the backslash sequence, including the backslash character.

DESCRIPTION

The use of Tcl_Backslash is deprecated in favor of Tcl_UtfBackslash.

This is a utility procedure provided for backwards compatibility with non–internationalized Tcl extensions. It parses a backslash sequence and returns the low byte of the Unicode character corresponding to the sequence. Tcl_Backslash modifies *countPtr to contain the number of characters in the backslash sequence.

See the Tcl manual entry for information on the valid backslash sequences. All of the sequences described in the Tcl manual entry are supported by Tcl_Backslash.

SEE ALSO

Tcl_Tcl_UtfBackslash

KEYWORDS

backslash, parse

Copyright ' 1989-1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
BoolObj

NAME

Tcl_NewBooleanObj, Tcl_SetBooleanObj, Tcl_GetBooleanFromObj – manipulate Tcl objects as boolean values

SYNOPSIS

#include <tcl.h>
Tcl_Obj *
Tcl_NewBooleanObj(boolValue)
Tcl_SetBooleanObj(objPtr, boolValue)
int
Tcl_GetBooleanFromObj(interp, objPtr, boolPtr)

ARGUMENTS

int boolValue (in)
    Integer value used to initialize or set a boolean object. If the integer is nonzero, the boolean object is
    set to 1; otherwise the boolean object is set to 0.
Tcl_Obj *objPtr (in/out)
    For Tcl_SetBooleanObj, this points to the object to be converted to boolean type. For
    Tcl_GetBooleanFromObj, this refers to the object from which to get a boolean value; if objPtr does
    not already point to a boolean object, an attempt will be made to convert it to one.
Tcl_Interp *interp (in/out)
    If an error occurs during conversion, an error message is left in the interpreter's result object unless
    interp is NULL.
int *boolPtr (out)
    Points to place where Tcl_GetBooleanFromObj stores the boolean value (0 or 1) obtained from
    objPtr.

DESCRIPTION

These procedures are used to create, modify, and read boolean Tcl objects from C code. Tcl_NewBooleanObj
and Tcl_SetBooleanObj will create a new object of boolean type or modify an existing object to have
boolean type. Both of these procedures set the object to have the boolean value (0 or 1) specified by
boolValue; if boolValue is nonzero, the object is set to 1, otherwise to 0. Tcl_NewBooleanObj returns a
pointer to a newly created object with reference count zero. Both procedures set the object's type to be
boolean and assign the boolean value to the object's internal representation longValue member.
Tcl_SetBooleanObj invalidates any old string representation and, if the object is not already a boolean object,
freees any old internal representation.
Tcl_GetBooleanFromObj attempts to return a boolean value from the Tcl object objPtr. If the object is not already a boolean object, it will attempt to convert it to one. If an error occurs during conversion, it returns TCL_ERROR and leaves an error message in the interpreter's result object unless interp is NULL. Otherwise, Tcl_GetBooleanFromObj returns TCL_OK and stores the boolean value in the address given by boolPtr. If the object is not already a boolean object, the conversion will free any old internal representation. Objects having a string representation equal to any of 0, false, no, or off have a boolean value 0; if the string representation is any of 1, true, yes, or on the boolean value is 1. Any of these string values may be abbreviated, and upper-case spellings are also acceptable.

SEE ALSO

Tcl_NewObj, Tcl_DecrRefCount, Tcl_IncrRefCount, Tcl_GetObjResult

KEYWORDS

boolean, boolean object, boolean type, internal representation, object, object type, string representation

Copyright ' 1996-1997 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
ByteArrObj

NAME

Tcl_NewByteArrayObj, Tcl_SetByteArrayObj, Tcl_GetByteArrayFromObj, Tcl_SetByteArrayLength – manipulate Tcl objects as a arrays of bytes

SYNOPSIS

#include <tcl.h>

Tcl_Obj *
Tcl_NewByteArrayObj(bytes, length)

void
Tcl_SetByteArrayObj(objPtr, bytes, length)

unsigned char *
Tcl_GetByteArrayFromObj(objPtr, lengthPtr)

unsigned char *
Tcl_SetByteArrayLength(objPtr, length)

ARGUMENTS

CONST unsigned char *bytes (in)

The array of bytes used to initialize or set a byte–array object.

int length (in)

The length of the array of bytes. It must be >= 0.

Tcl_Obj *objPtr (in/out)

For Tcl_SetByteArrayObj, this points to the object to be converted to byte–array type. For Tcl_GetByteArrayFromObj and Tcl_SetByteArrayLength, this points to the object from which to get the byte–array value; if objPtr does not already point to a byte–array object, it will be converted to one.

int *lengthPtr (out)

If non–NULL, filled with the length of the array of bytes in the object.

DESCRIPTION

These procedures are used to create, modify, and read Tcl byte–array objects from C code. Byte–array objects are typically used to hold the results of binary IO operations or data structures created with the binary command. In Tcl, an array of bytes is not equivalent to a string. Conceptually, a string is an array of Unicode characters, while a byte–array is an array of 8–bit quantities with no implicit meaning. Accesser functions are provided to get the string representation of a byte–array or to convert an arbitrary object to a byte–array. Obtaining the string representation of a byte–array object (by calling Tcl_GetStringFromObj) produces a properly formed UTF–8 sequence with a one–to–one mapping between the bytes in the internal representation and the UTF–8 characters in the string representation.
**Tcl_NewByteArrayObj** and **Tcl_SetByteArrayObj** will create a new object of byte–array type or modify an existing object to have a byte–array type. Both of these procedures set the object's type to be byte–array and set the object's internal representation to a copy of the array of bytes given by `bytes`. **Tcl_NewByteArrayObj** returns a pointer to a newly allocated object with a reference count of zero. **Tcl_SetByteArrayObj** invalidates any old string representation and, if the object is not already a byte–array object, frees any old internal representation.

**Tcl_GetByteArrayFromObj** converts a Tcl object to byte–array type and returns a pointer to the object's new internal representation as an array of bytes. The length of this array is stored in `lengthPtr` if `lengthPtr` is non–NULL. The storage for the array of bytes is owned by the object and should not be freed. The contents of the array may be modified by the caller only if the object is not shared and the caller invalidates the string representation.

**Tcl_SetByteArrayLength** converts the Tcl object to byte–array type and changes the length of the object's internal representation as an array of bytes. If `length` is greater than the space currently allocated for the array, the array is reallocated to the new length; the newly allocated bytes at the end of the array have arbitrary values. If `length` is less than the space currently allocated for the array, the length of array is reduced to the new length. The return value is a pointer to the object's new array of bytes.

**SEE ALSO**

**Tcl_GetStringFromObj**, **Tcl_NewObj**, **Tcl_IncrRefCount**, **Tcl_DecrRefCount**

**KEYWORDS**

object, byte array, utf, unicode, internationalization

*Copyright ’ 1997 Sun Microsystems, Inc.*
CallDel

NAME

Tcl_CallWhenDeleted, Tcl_DontCallWhenDeleted – Arrange for callback when interpreter is deleted

SYNOPSIS

#include <tcl.h>

Tcl_CallWhenDeleted(interp, proc, clientData)
Tcl_DontCallWhenDeleted(interp, proc, clientData)

ARGUMENTS

Tcl_Interp *interp (in)
    Interpreter with which to associated callback.
Tcl_InterpDeleteProc *proc (in)
    Procedure to call when interp is deleted.
ClientData clientData (in)
    Arbitrary one-word value to pass to proc.

DESCRIPTION

Tcl_CallWhenDeleted arranges for proc to be called by Tcl_DeleteInterp if/when interp is deleted at some future time. Proc will be invoked just before the interpreter is deleted, but the interpreter will still be valid at the time of the call. Proc should have arguments and result that match the type Tcl_InterpDeleteProc:

typedef void Tcl_InterpDeleteProc(
    ClientData, 
    Tcl_Interp *interp);

The clientData and interp parameters are copies of the clientData and interp arguments given to Tcl_CallWhenDeleted. Typically, clientData points to an application-specific data structure that proc uses to perform cleanup when an interpreter is about to go away. Proc does not return a value.

Tcl_DontCallWhenDeleted cancels a previous call to Tcl_CallWhenDeleted with the same arguments, so that proc won’t be called after all when interp is deleted. If there is no deletion callback that matches interp, proc, and clientData then the call to Tcl_DontCallWhenDeleted has no effect.

KEYWORDS

callback, delete, interpreter

Copyright ' 1993 The Regents of the University of California. 
Copyright ' 1994-1996 Sun Microsystems, Inc.
CmdCmplt

NAME

Tcl_CommandComplete – Check for unmatched braces in a Tcl command

SYNOPSIS

#include <tcl.h>

int
Tcl_CommandComplete(cmd)

ARGUMENTS

CONST char *cmd (in)
Command string to test for completeness.

DESCRIPTION

Tcl_CommandComplete takes a Tcl command string as argument and determines whether it contains one or more complete commands (i.e. there are no unclosed quotes, braces, brackets, or variable references). If the command string is complete then it returns 1; otherwise it returns 0.

KEYWORDS

complete command, partial command

Copyright ' 1989-1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
Concat

NAME

Tcl_Concat – concatenate a collection of strings

SYNOPSIS

#include <tcl.h>
CONST char *
Tcl_Concat(argc, argv)

ARGUMENTS

int argc (in)
   Number of strings.
CONST char * CONST argv[] (in)
   Array of strings to concatenate. Must have argc entries.

DESCRIPTION

Tcl_Concat is a utility procedure used by several of the Tcl commands. Given a collection of strings, it
concatenates them together into a single string, with the original strings separated by spaces. This procedure
behaves differently than Tcl_Merge, in that the arguments are simply concatenated: no effort is made to
ensure proper list structure. However, in most common usage the arguments will all be proper lists
themselves; if this is true, then the result will also have proper list structure.

Tcl_Concat eliminates leading and trailing white space as it copies strings from argv to the result. If an
element of argv consists of nothing but white space, then that string is ignored entirely. This white−space
removal was added to make the output of the concat command cleaner−looking.

The result string is dynamically allocated using Tcl_Alloc; the caller must eventually release the space by
calling Tcl_Free.

SEE ALSO

Tcl_ConcatObj

KEYWORDS

concatenate, strings

Copyright ' 1989−1993 The Regents of the University of California.
Copyright ' 1994−1996 Sun Microsystems, Inc.
CrtChannel

NAME

Tcl_CreateChannel, Tcl_GetChannelInstanceData, Tcl_GetChannelType, Tcl_GetChannelName, Tcl_GetChannelHandle, Tcl_GetChannelMode, Tcl_GetChannelBufferSize, Tcl_SetChannelBufferSize, Tcl_NotifyChannel, Tcl_BadChannelOption, Tcl_ChannelName, Tcl_ChannelVersion, Tcl_ChannelBlockModeProc, Tcl_ChannelCloseProc, Tcl_ChannelClose2Proc, Tcl_ChannelInputProc, Tcl_ChannelOutputProc, Tcl_ChannelSeekProc, Tcl_ChannelWideSeekProc, Tcl_ChannelSetOptionProc, Tcl_ChannelGetOptionProc, Tcl_ChannelWatchProc, Tcl_ChannelGetHandleProc, Tcl_ChannelFlushProc, Tcl_ChannelHandlerProc, Tcl_IsChannelShared, Tcl_IsChannelRegistered, Tcl_CutChannel, Tcl_SpliceChannel, Tcl_IsChannelExisting, Tcl_ClearChannelHandlers, Tcl_GetChannelThread, Tcl_ChannelBuffered – procedures for creating and manipulating channels

SYNOPSIS
ARGUMENTS
DESCRIPTION
TCL_CHANNELTYPE
TYPENAME
VERSION
BLOCKMODEPROC
CLOSEPROC AND CLOSE2PROC
INPUTPROC
OUTPUTPROC
SEEKPROC AND WIDESEEKPROC
SETOPTIONPROC
GETOPTIONPROC
WATCHPROC
GETHANDLEPROC
FLUSHPROC
HANDLERPROC
TCL_BADCHANNELOPTION
OLD_CHANNEL_TYPES
SEE ALSO
KEYWORDS

NAME

Tcl_CreateChannel, Tcl_GetChannelInstanceData, Tcl_GetChannelType, Tcl_GetChannelName, Tcl_GetChannelHandle, Tcl_GetChannelMode, Tcl_GetChannelBufferSize, Tcl_SetChannelBufferSize, Tcl_NotifyChannel, Tcl_BadChannelOption, Tcl_ChannelName, Tcl_ChannelVersion,
SYNOPSIS

#include <tcl.h>

Tcl_CreateChannel(typePtr, channelName, instanceData, mask)
ClientData
Tcl_GetChannelInstanceData(channel)
Tcl_ChannelType *
Tcl_GetChannelType(channel)
CONST char *
Tcl_GetChannelName(channel)

int
Tcl_GetChannelHandle(channel, direction, handlePtr)
TclThreadId
Tcl_GetChannelThread(channel)

int
Tcl_GetChannelMode(channel)

int
Tcl_GetChannelBufferSize(channel)
Tcl_SetChannelBufferSize(channel, size)
Tcl_NotifyChannel(channel, mask)

int
Tcl_BadChannelOption(interp, optionName, optionList)

int
Tcl_IsChannelShared(channel)

int
Tcl_IsChannelRegistered(interp, channel)

int
Tcl_IsChannelExisting(channelName)

void
Tcl_CutChannel(channel)

void
Tcl_SpliceChannel(channel)

void
Tcl_ClearChannelHandlers(channel)

int
Tcl_ChannelBuffered(channel)
CONST char *
**Tcl_ChannelName(typePtr)**
Tcl_ChannelTypeVersion

**Tcl_ChannelVersion(typePtr)**
Tcl_DriverBlockModeProc *

**Tcl_ChannelBlockModeProc(typePtr)**
Tcl_DriverCloseProc *

**Tcl_ChannelCloseProc(typePtr)**
Tcl_DriverClose2Proc *

**Tcl_ChannelClose2Proc(typePtr)**
Tcl_DriverInputProc *

**Tcl_ChannelInputProc(typePtr)**
Tcl_DriverOutputProc *

**Tcl_ChannelOutputProc(typePtr)**
Tcl_DriverSeekProc *

**Tcl_ChannelSeekProc(typePtr)**
Tcl_DriverWideSeekProc *

**Tcl_ChannelWideSeekProc(typePtr)**
Tcl_DriverSetOptionProc *

**Tcl_ChannelSetOptionProc(typePtr)**
Tcl_DriverGetOptionProc *

**Tcl_ChannelGetOptionProc(typePtr)**
Tcl_DriverWatchProc *

**Tcl_ChannelWatchProc(typePtr)**
Tcl_DriverGetHandleProc *

**Tcl_ChannelGetHandleProc(typePtr)**
Tcl_DriverFlushProc *

**Tcl_ChannelFlushProc(typePtr)**
Tcl_DriverHandlerProc *

**Tcl_ChannelHandlerProc(typePtr)**

ARGUMENTS

**Tcl(ChannelType **typePtr** (in))**
Points to a structure containing the addresses of procedures that can be called to perform I/O and other functions on the channel.

**CONST char **channelName** (in)**
The name of this channel, such as file3; must not be in use by any other channel. Can be NULL, in which case the channel is created without a name.

**ClientData instanceData** (in)
Arbitrary one-word value to be associated with this channel. This value is passed to procedures in typePtr when they are invoked.

**int mask** (in)
OR-ed combination of TCL_READABLE and TCLWRITABLE to indicate whether a channel is readable and writable.

**Tcl_Channel channel** (in)
The channel to operate on.

```
int direction (in)
TCL_READABLE means the input handle is wanted; TCL_WRITABLE means the output handle is wanted.
```

```
ClientData *handlePtr (out)
Points to the location where the desired OS-specific handle should be stored.
```

```
int size (in)
The size, in bytes, of buffers to allocate in this channel.
```

```
int mask (in)
An OR-ed combination of TCL_READABLE, TCL_WRITABLE and TCL_EXCEPTION that indicates events that have occurred on this channel.
```

```
Tcl_Interp *interp (in)
Current interpreter. (can be NULL)
```

```
CONST char *optionName (in)
Name of the invalid option.
```

```
CONST char *optionList (in)
Specific options list (space separated words, without "−") to append to the standard generic options list. Can be NULL for generic options error message only.
```

## DESCRIPTION

Tcl uses a two-layered channel architecture. It provides a generic upper layer to enable C and Tcl programs to perform input and output using the same APIs for a variety of files, devices, sockets etc. The generic C APIs are described in the manual entry for Tcl_OpenFileChannel.

The lower layer provides type-specific channel drivers for each type of device supported on each platform. This manual entry describes the type-specific channel drivers. It also explains how new types of channels can be added by providing new channel drivers.

Channel drivers consist of a number of components: First, each channel driver provides a Tcl_ChannelType structure containing pointers to functions implementing the various operations used by the generic layer to communicate with the channel driver. The Tcl_ChannelType structure and the functions referenced by it are described in the section TCL_CHANNELTYPE, below.

Second, channel drivers usually provide a Tcl command to create instances of that type of channel. For example, the Tcl open command creates channels that use the file and command channel drivers, and the Tcl socket command creates channels that use TCP sockets for network communication.

Third, a channel driver optionally provides a C function to open channel instances of that type. For example, Tcl_OpenFileChannel opens a channel that uses the file channel driver, and Tcl_OpenTcpClient opens a channel that uses the TCP network protocol. These creation functions typically use Tcl_CreateChannel internally to open the channel.

To add a new type of channel you must implement a C API or a Tcl command that opens a channel by invoking Tcl_CreateChannel. When your driver calls Tcl_CreateChannel it passes in a Tcl_ChannelType
structure describing the driver's I/O procedures. The generic layer will then invoke the functions referenced in that structure to perform operations on the channel.

**Tcl_CreateChannel** opens a new channel and associates the supplied `typePtr` and `instanceData` with it. The channel is opened in the mode indicated by `mask`. For a discussion of channel drivers, their operations and the `Tcl_ChannelType` structure, see the section TCL_CHANNELTYPE, below.

**Tcl_CreateChannel** interacts with the code managing the standard channels. Once a standard channel was initialized either through a call to **Tcl_GetStdChannel** or a call to **Tcl_SetStdChannel** closing this standard channel will cause the next call to **Tcl_CreateChannel** to make the new channel the new standard channel too. See **Tcl_StandardChannels** for a general treatise about standard channels and the behaviour of the Tcl library with regard to them.

**Tcl_GetChannelInstanceData** returns the instance data associated with the channel in `channel`. This is the same as the `instanceData` argument in the call to **Tcl_CreateChannel** that created this channel.

**Tcl_GetChannelType** returns a pointer to the `Tcl_ChannelType` structure used by the channel in the `channel` argument. This is the same as the `typePtr` argument in the call to **Tcl_CreateChannel** that created this channel.

**Tcl_GetChannelName** returns a string containing the name associated with the channel, or NULL if the `channelName` argument to **Tcl_CreateChannel** was NULL.

**Tcl_GetChannelHandle** places the OS-specific device handle associated with `channel` for the given `direction` in the location specified by `handlePtr` and returns TCL_OK. If the channel does not have a device handle for the specified direction, then TCL_ERROR is returned instead. Different channel drivers will return different types of handle. Refer to the manual entries for each driver to determine what type of handle is returned.

**Tcl_GetChannelThread** returns the id of the thread currently managing the specified `channel`. This allows channel drivers to send their file events to the correct event queue even for a multi-threaded core.

**Tcl_GetChannelMode** returns an OR-ed combination of TCL_READABLE and TCL_WRITABLE, indicating whether the channel is open for input and output.

**Tcl_GetChannelBufferSize** returns the size, in bytes, of buffers allocated to store input or output in `channel`. If the value was not set by a previous call to **Tcl_SetChannelBufferSize**, described below, then the default value of 4096 is returned.

**Tcl_SetChannelBufferSize** sets the size, in bytes, of buffers that will be allocated in subsequent operations on the channel to store input or output. The `size` argument should be between ten and one million, allowing buffers of ten bytes to one million bytes. If `size` is outside this range, **Tcl_SetChannelBufferSize** sets the buffer size to 4096.

**Tcl_NotifyChannel** is called by a channel driver to indicate to the generic layer that the events specified by
mask have occurred on the channel. Channel drivers are responsible for invoking this function whenever the channel handlers need to be called for the channel. See WATCHPROC below for more details.

Tcl_BadChannelOption is called from driver specific set or get option procs to generate a complete error message.

Tcl_ChannelBuffered returns the number of bytes of input currently buffered in the internal buffer (push back area) of the channel itself. It does not report about the data in the overall buffers for the stack of channels the supplied channel is part of.

Tcl_IsChannelShared checks the refcount of the specified channel and returns whether the channel was shared among multiple interpreters (result == 1) or not (result == 0).

Tcl_IsChannelRegistered checks whether the specified channel is registered in the given interpreter (result == 1) or not (result == 0).

Tcl_IsChannelExisting checks whether a channel with the specified name is registered in the (thread)−global list of all channels (result == 1) or not (result == 0).

Tcl_CutChannel removes the specified channel from the (thread)−global list of all channels (of the current thread). Application to a channel still registered in some interpreter is not allowed.

Tcl_SpliceChannel adds the specified channel to the (thread)−global list of all channels (of the current thread). Application to a channel registered in some interpreter is not allowed.

Tcl_ClearChannelHandlers removes all channelhandlers and event scripts associated with the specified channel, thus shutting down all event processing for this channel.

TCL_CHANNELTYPE

A channel driver provides a Tcl_ChannelType structure that contains pointers to functions that implement the various operations on a channel; these operations are invoked as needed by the generic layer. The structure was versioned starting in Tcl 8.3.2/8.4 to correct a problem with stacked channel drivers. See the OLD CHANNEL TYPES section below for details about the old structure.

The Tcl_ChannelType structure contains the following fields:

typedef struct Tcl_ChannelType {
    char *typeName;
    Tcl_ChannelTypeVersion;
    Tcl_DriverCloseProc *closeProc;
    Tcl_DriverInputProc *inputProc;
    Tcl_DriverOutputProc *outputProc;
    Tcl_DriverSeekProc *seekProc;
    Tcl_DriverSetOptionProc *setOptionProc;
    Tcl_DriverGetOptionProc *getOptionProc;
    Tcl_DriverWatchProc *watchProc;
}
The driver must provide implementations for all functions except `blockModeProc`, `seekProc`, `setOptionProc`, `getOptionProc`, and `close2Proc`, which may be specified as NULL. Other functions that cannot be implemented for this type of device should return `EINVAL` when invoked to indicate that they are not implemented, except in the case of `flushProc` and `handlerProc`, which should be specified as NULL if not otherwise defined.

The user should only use the above structure for `Tcl_ChannelType` instantiation. When referencing fields in a `Tcl_ChannelType` structure, the following functions should be used to obtain the values:

- `Tcl_ChannelName`
- `Tcl_ChannelVersion`
- `Tcl_ChannelBlockModeProc`
- `Tcl_ChannelCloseProc`
- `Tcl_ChannelClose2Proc`
- `Tcl_ChannelInputProc`
- `Tcl_ChannelOutputProc`
- `Tcl_ChannelSeekProc`
- `Tcl_ChannelWideSeekProc`
- `Tcl_ChannelSetOptionProc`
- `Tcl_ChannelGetOptionProc`
- `Tcl_ChannelWatchProc`
- `Tcl_ChannelGetHandleProc`
- `Tcl_ChannelFlushProc`
- `Tcl_ChannelHandlerProc`

The change to the structures was made in such a way that standard channel types are binary compatible. However, channel types that use stacked channels (i.e., TLS, Trf) have new versions to correspond to the above change since the previous code for stacked channels had problems.

**TYPENAME**

The `typeName` field contains a null-terminated string that identifies the type of the device implemented by this driver, e.g., `file` or `socket`.

This value can be retrieved with `Tcl_ChannelName`, which returns a pointer to the string.

**VERSION**

The `version` field should be set to `TCL_CHANNEL_VERSION_2`. If it is not set to this value `TCL_CHANNEL_VERSION_3`, then this `Tcl_ChannelType` is assumed to have the older structure. See `OLD CHANNEL TYPES` for more details. While Tcl will recognize and function with either structure, stacked channels must be of at least `TCL_CHANNEL_VERSION_2` to function correctly.

This value can be retrieved with `Tcl_ChannelVersion`, which returns one of `TCL_CHANNEL_VERSION_3`, `TCL_CHANNEL_VERSION_2` or `TCL_CHANNEL_VERSION_1`. 
**BLOCKMODEPROC**

The `blockModeProc` field contains the address of a function called by the generic layer to set blocking and nonblocking mode on the device. `BlockModeProc` should match the following prototype:

```c
typedef int Tcl_DriverBlockModeProc(
    ClientData instanceData,
    int mode);
```

The `instanceData` is the same as the value passed to `Tcl_CreateChannel` when this channel was created. The `mode` argument is either `TCL_MODE_BLOCKING` or `TCL_MODE_NONBLOCKING` to set the device into blocking or nonblocking mode. The function should return zero if the operation was successful, or a nonzero POSIX error code if the operation failed.

If the operation is successful, the function can modify the supplied `instanceData` to record that the channel entered blocking or nonblocking mode and to implement the blocking or nonblocking behavior. For some device types, the blocking and nonblocking behavior can be implemented by the underlying operating system; for other device types, the behavior must be emulated in the channel driver.

This value can be retrieved with `Tcl_ChannelBlockModeProc`, which returns a pointer to the function.

A channel driver not supplying a `blockModeProc` has to be very, very careful. It has to tell the generic layer exactly which blocking mode is acceptable to it, and should this also document for the user so that the blocking mode of the channel is not changed to an unacceptable value. Any confusion here may lead the interpreter into a (spurious and difficult to find) deadlock.

**CLOSEPROC AND CLOSE2PROC**

The `closeProc` field contains the address of a function called by the generic layer to clean up driver–related information when the channel is closed. `CloseProc` must match the following prototype:

```c
typedef int Tcl_DriverCloseProc(
    ClientData instanceData,
    Tcl_Interp *interp);
```

The `instanceData` argument is the same as the value provided to `Tcl_CreateChannel` when the channel was created. The function should release any storage maintained by the channel driver for this channel, and close the input and output devices encapsulated by this channel. All queued output will have been flushed to the device before this function is called, and no further driver operations will be invoked on this instance after calling the `closeProc`. If the close operation is successful, the procedure should return zero; otherwise it should return a nonzero POSIX error code. In addition, if an error occurs and `interp` is not NULL, the procedure should store an error message in the interpreter's result.

Alternatively, channels that support closing the read and write sides independently may set `closeProc` to `TCL_CLOSE2PROC` and set `close2Proc` to the address of a function that matches the following prototype:
typedef int Tcl_DriverClose2Proc(
    ClientData instanceData,
    Tcl_Interp *interp,
    flags);

The close2Proc will be called with flags set to an OR'ed combination of TCL_CLOSE_READ or TCL_CLOSE_WRITE to indicate that the driver should close the read and/or write side of the channel. The channel driver may be invoked to perform additional operations on the channel after close2Proc is called to close one or both sides of the channel. If flags is 0 (zero), the driver should close the channel in the manner described above for closeProc. No further operations will be invoked on this instance after close2Proc is called with all flags cleared. In all cases, the close2Proc function should return zero if the close operation was successful; otherwise it should return a nonzero POSIX error code. In addition, if an error occurs and interp is not NULL, the procedure should store an error message in the interpreter's result.

These value can be retrieved with Tcl_ChannelCloseProc or Tcl_ChannelClose2Proc, which returns a pointer to the respective function.

INPUTPROC

The inputProc field contains the address of a function called by the generic layer to read data from the file or device and store it in an internal buffer. InputProc must match the following prototype:

typedef int Tcl_DriverInputProc(
    ClientData instanceData,
    char *buf,
    int bufSize,
    int *errorCodePtr);

InstanceData is the same as the value passed to Tcl_CreateChannel when the channel was created. The buf argument points to an array of bytes in which to store input from the device, and the bufSize argument indicates how many bytes are available at buf.

The errorCodePtr argument points to an integer variable provided by the generic layer. If an error occurs, the function should set the variable to a POSIX error code that identifies the error that occurred.

The function should read data from the input device encapsulated by the channel and store it at buf. On success, the function should return a nonnegative integer indicating how many bytes were read from the input device and stored at buf. On error, the function should return −1. If an error occurs after some data has been read from the device, that data is lost.

If inputProc can determine that the input device has some data available but less than requested by the bufSize argument, the function should only attempt to read as much data as is available and return without blocking. If the input device has no data available whatsoever and the channel is in nonblocking mode, the function should return an EAGAIN error. If the input device has no data available whatsoever and the channel is in blocking mode, the function should block for the shortest possible time until at least one byte of data can be read from the device; then, it should return as much data as it can read without blocking.
This value can be retrieved with `Tcl_ChannelInputProc`, which returns a pointer to the function.

**OUTPUTPROC**

The `outputProc` field contains the address of a function called by the generic layer to transfer data from an internal buffer to the output device. `OutputProc` must match the following prototype:

```c
typedef int Tcl_DriverOutputProc(
    ClientData instanceData,
    CONStBChar * buf,
    int toWrite,
    int * errorCodePtr);
```

*InstanceData* is the same as the value passed to `Tcl_CreateChannel` when the channel was created. The *buf* argument contains an array of bytes to be written to the device, and the *toWrite* argument indicates how many bytes are to be written from the *buf* argument.

The `errorCodePtr` argument points to an integer variable provided by the generic layer. If an error occurs, the function should set this variable to a POSIX error code that identifies the error.

The function should write the data at *buf* to the output device encapsulated by the channel. On success, the function should return a nonnegative integer indicating how many bytes were written to the output device. The return value is normally the same as *toWrite*, but may be less in some cases such as if the output operation is interrupted by a signal. If an error occurs the function should return −1. In case of error, some data may have been written to the device.

If the channel is nonblocking and the output device is unable to absorb any data whatsoever, the function should return −1 with an **EAGAIN** error without writing any data.

This value can be retrieved with `Tcl_ChannelOutputProc`, which returns a pointer to the function.

**SEEKPROC AND WIDESEEKPROC**

The `seekProc` field contains the address of a function called by the generic layer to move the access point at which subsequent input or output operations will be applied. `SeekProc` must match the following prototype:

```c
typedef int Tcl_DriverSeekProc(
    ClientData instanceData,
    offlong offset,
    int seekMode,
    int * errorCodePtr);
```

*InstanceData* argument is the same as the value given to `Tcl_CreateChannel` when this channel was created. *Offset* and *seekMode* have the same meaning as for the `Tcl_Seqek` procedure (described in the manual entry for `Tcl_OpenFileChannel`).
The `errorCodePtr` argument points to an integer variable provided by the generic layer for returning `errno` values from the function. The function should set this variable to a POSIX error code if an error occurs. The function should store an `EINVAL` error code if the channel type does not implement seeking.

The return value is the new access point or −1 in case of error. If an error occurred, the function should not move the access point.

If there is a non-NULL `seekProc` field, the `wideSeekProc` field may contain the address of an alternative function to use which handles wide (i.e. larger than 32-bit) offsets, so allowing seeks within files larger than 2GB. The `wideSeekProc` will be called in preference to the `seekProc`, but both must be defined if the `wideSeekProc` is defined. `WideSeekProc` must match the following prototype:

```c
typedef Tcl_WideInt Tcl_DriverWideSeekProc(
    ClientData instanceData,
    Tcl_WideInt offset,
    int seekMode,
    int *errorCodePtr);
```

The arguments and return values mean the same thing as with `seekProc` above, except that the type of offsets and the return type are different.

The `seekProc` value can be retrieved with `Tcl_ChannelSeekProc`, which returns a pointer to the function, and similarly the `wideSeekProc` can be retrieved with `Tcl_ChannelWideSeekProc`.

### SETOPTIONPROC

The `setOptionProc` field contains the address of a function called by the generic layer to set a channel type specific option on a channel. `setOptionProc` must match the following prototype:

```c
typedef int Tcl_DriverSetOptionProc(
    ClientData instanceData,
    Tcl_Interp *interp,
    const char *optionName,
    const char *newValue);
```

`optionName` is the name of an option to set, and `newValue` is the new value for that option, as a string. The `instanceData` is the same as the value given to `Tcl_CreateChannel` when this channel was created. The function should do whatever channel type specific action is required to implement the new value of the option.

Some options are handled by the generic code and this function is never called to set them, e.g. `−blockmode`. Other options are specific to each channel type and the `setOptionProc` procedure of the channel driver will get called to implement them. The `setOptionProc` field can be NULL, which indicates that this channel type supports no type specific options.

If the option value is successfully modified to the new value, the function returns `TCL_OK`. It should call `Tcl_BadChannelOption` which itself returns `TCL_ERROR` if the `optionName` is unrecognized. If `newValue` specifies a value for the option that is not supported or if a system call error occurs, the function should leave
an error message in the result field of interp if interp is not NULL. The function should also call Tcl_SetErrno to store an appropriate POSIX error code.

This value can be retrieved with Tcl_ChannelSetOptionProc, which returns a pointer to the function.

**GETOPTIONPROC**

The getOptionProc field contains the address of a function called by the generic layer to get the value of a channel type specific option on a channel. getOptionProc must match the following prototype:

```c
typedef int Tcl_DriverGetOptionProc(
    ClientData instanceData,
    Tcl_Interp *interp,
    CONST char *optionName,
    Tcl_DString *optionValue);
```

*OptionName* is the name of an option supported by this type of channel. If the option name is not NULL, the function stores its current value, as a string, in the Tcl dynamic string *optionValue*. If *optionName* is NULL, the function stores in *optionValue* an alternating list of all supported options and their current values. On success, the function returns TCL_OK. It should call Tcl_BadChannelOption which itself returns TCL_ERROR if the *optionName* is unrecognized. If a system call error occurs, the function should leave an error message in the result of interp if interp is not NULL. The function should also call Tcl_SetErrno to store an appropriate POSIX error code.

Some options are handled by the generic code and this function is never called to retrieve their value, e.g. −blockmode. Other options are specific to each channel type and the getOptionProc procedure of the channel driver will get called to implement them. The getOptionProc field can be NULL, which indicates that this channel type supports no type specific options.

This value can be retrieved with Tcl_ChannelGetOptionProc, which returns a pointer to the function.

**WATCHPROC**

The watchProc field contains the address of a function called by the generic layer to initialize the event notification mechanism to notice events of interest on this channel. WatchProc should match the following prototype:

```c
typedef void Tcl_DriverWatchProc(
    ClientData instanceData,
    int mask);
```

The *instanceData* is the same as the value passed to Tcl_CreateChannel when this channel was created. The *mask* argument is an OR−ed combination of TCL_READABLE, TCL_WRITABLE and TCL_EXCEPTION; it indicates events the caller is interested in noticing on this channel.

The function should initialize device type specific mechanisms to notice when an event of interest is present
on the channel. When one or more of the designated events occurs on the channel, the channel driver is responsible for calling TclNotifyChannel to inform the generic channel module. The driver should take care not to starve other channel drivers or sources of callbacks by invoking TclNotifyChannel too frequently. Fairness can be insured by using the Tcl event queue to allow the channel event to be scheduled in sequence with other events. See the description of TclQueueEvent for details on how to queue an event.

This value can be retrieved with Tcl_ChannelWatchProc, which returns a pointer to the function.

**GETHANDLEPROC**

The `getHandleProc` field contains the address of a function called by the generic layer to retrieve a device-specific handle from the channel. `GetHandleProc` should match the following prototype:

```c
typedef int Tcl_DriverGetHandleProc(
    ClientData instanceData,
    int direction,
    ClientData *handlePtr);
```

`InstanceData` is the same as the value passed to TclCreateChannel when this channel was created. The `direction` argument is either TCL_READABLE to retrieve the handle used for input, or TCL_WRITABLE to retrieve the handle used for output.

If the channel implementation has device-specific handles, the function should retrieve the appropriate handle associated with the channel, according the `direction` argument. The handle should be stored in the location referred to by `handlePtr`, and TCL_OK should be returned. If the channel is not open for the specified direction, or if the channel implementation does not use device handles, the function should return TCL_ERROR.

This value can be retrieved with Tcl_ChannelGetHandleProc, which returns a pointer to the function.

**FLUSHPROC**

The `flushProc` field is currently reserved for future use. It should be set to NULL. `FlushProc` should match the following prototype:

```c
typedef int Tcl_DriverFlushProc(
    ClientData instanceData);
```

This value can be retrieved with Tcl_ChannelFlushProc, which returns a pointer to the function.

**HANDLERPROC**

The `handlerProc` field contains the address of a function called by the generic layer to notify the channel that an event occurred. It should be defined for stacked channel drivers that wish to be notified of events that occur on the underlying (stacked) channel. `HandlerProc` should match the following prototype:
typedef int Tcl_DriverHandlerProc(
    ClientData instanceData,
    interestMask);

InstanceData is the same as the value passed to Tcl_CreateChannel when this channel was created. The interestMask is an OR-ed combination of TCL_READABLE or TCL_WRITABLE; it indicates what type of event occurred on this channel.

This value can be retrieved with Tcl_ChannelHandlerProc, which returns a pointer to the function.

TCL_BADCHANNELOPTION

This procedure generates a "bad option" error message in an (optional) interpreter. It is used by channel drivers when a invalid Set/Get option is requested. Its purpose is to concatenate the generic options list to the specific ones and factorize the generic options error message string.

It always return TCL_ERROR

An error message is generated in interp's result object to indicate that a command was invoked with the a bad option. The message has the form

bad option "blah": should be one of
    <...generic options...>+<...specific options...>
so you get for instance:
    bad option "−blah": should be one of −blocking,
    −buffering, −buffersize, −eofchar, −translation,
    −peername, or −sockname
when called with optionList="peername sockname"

``blah'' is the optionName argument and ``<specific options>'' is a space separated list of specific option words. The function takes good care of inserting minus signs before each option, commas after, and an "or" before the last option.

OLD CHANNEL TYPES

The original (8.3.1 and below) Tcl_ChannelType structure contains the following fields:

typedef struct Tcl_ChannelType {
    typeName;
    Tcl_DriverBlockModeProc;
    Tcl_DriverCloseProc;
    Tcl_DriverCloseErr;
    Tcl_DriverInputProc;
    Tcl_DriverOutputProc;
    Tcl_DriverSeekProc;
    Tcl_DriverSetOptionProc;
    Tcl_DriverGetOptionProc;
    Tcl_DriverWatchProc;
    Tcl_DriverGetHandleProc;
    Tcl_DriverClose2Proc;
}
It is still possible to create channel with the above structure. The internal channel code will determine the version. It is imperative to use the new Tcl_ChannelType structure if you are creating a stacked channel driver, due to problems with the earlier stacked channel implementation (in 8.2.0 to 8.3.1).

Prior to 8.4.0 (i.e. during the later releases of 8.3 and early part of the 8.4 development cycle) the Tcl_ChannelType structure contained the following fields:

```c
typedef struct Tcl_ChannelType {
    char *typeName;
    Tcl_ChannelTypeVersion version;
    Tcl_DriverCloseProc *closeProc;
    Tcl_DriverInputProc *inputProc;
    Tcl_DriverOutputProc *outputProc;
    Tcl_DriverSeekProc *seekProc;
    Tcl_DriverSetOptionProc *setOptionProc;
    Tcl_DriverGetOptionProc *getOptionProc;
    Tcl_DriverWatchProc *watchProc;
    Tcl_DriverGetHandleProc *getHandleProc;
    Tcl_DriverClose2Proc *close2Proc;
    Tcl_DriverBlockModeProc *blockModeProc;
    Tcl_DriverFlushProc *flushProc;
    Tcl_DriverHandlerProc *handlerProc;
} Tcl_ChannelType;
```

When the above structure is registered as a channel type, the version field should always be TCL_CHANNEL_VERSION_2.

SEE ALSO

Tcl_Close, Tcl_OpenFileChannel, Tcl_SetErrno, Tcl_QueueEvent, Tcl_StackChannel, Tcl_GetStdChannel

KEYWORDS

blocking, channel driver, channel registration, channel type, nonblocking

Copyright ' 1996-1997 Sun Microsystems, Inc.
Copyright ' 1997-2000 Ajuba Solutions.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
CrtChnlHdlr

NAME

Tcl_CreateChannelHandler, Tcl_DeleteChannelHandler – call a procedure when a channel becomes readable or writable

SYNOPSIS

#include <tcl.h>

void
Tcl_CreateChannelHandler(channel, mask, proc, clientData)

void
Tcl_DeleteChannelHandler(channel, proc, clientData)

ARGUMENTS

Tcl_Channel channel (in)
    Tcl channel such as returned by Tcl_CreateChannel.

int mask (in)
    Conditions under which proc should be called: OR-ed combination of TCL_READABLE, TCL_WRITABLE and TCL_EXCEPTION. Specify a zero value to temporarily disable an existing handler.

Tcl_FileProc *proc (in)
    Procedure to invoke whenever the channel indicated by channel meets the conditions specified by mask.

ClientData clientData (in)
    Arbitrary one-word value to pass to proc.

DESCRIPTION

Tcl_CreateChannelHandler arranges for proc to be called in the future whenever input or output becomes possible on the channel identified by channel, or whenever an exceptional condition exists for channel. The conditions of interest under which proc will be invoked are specified by the mask argument. See the manual entry for fileevent for a precise description of what it means for a channel to be readable or writable. Proc must conform to the following prototype:

typedef void Tcl_ChannelProc(
    ClientData clientData,
    int mask
)

The clientData argument is the same as the value passed to Tcl_CreateChannelHandler when the handler was created. Typically, clientData points to a data structure containing application-specific information about the channel. Mask is an integer mask indicating which of the requested conditions actually exists for the channel; it will contain a subset of the bits from the mask argument to Tcl_CreateChannelHandler when the
Each channel handler is identified by a unique combination of *channel*, *proc* and *clientData*. There may be many handlers for a given channel as long as they don't have the same *channel*, *proc*, and *clientData*. If *Tcl_CreateChannelHandler* is invoked when there is already a handler for *channel*, *proc*, and *clientData*, then no new handler is created; instead, the *mask* is changed for the existing handler.

*Tcl_DeleteChannelHandler* deletes a channel handler identified by *channel*, *proc* and *clientData*; if no such handler exists, the call has no effect.

Channel handlers are invoked via the Tcl event mechanism, so they are only useful in applications that are event–driven. Note also that the conditions specified in the *mask* argument to *proc* may no longer exist when *proc* is invoked: for example, if there are two handlers for *TCL_READABLE* on the same channel, the first handler could consume all of the available input so that the channel is no longer readable when the second handler is invoked. For this reason it may be useful to use nonblocking I/O on channels for which there are event handlers.

**SEE ALSO**

*Notifier*, *Tcl_CreateChannel*, *Tcl_OpenFileChannel*, *vwait(n)*.

**KEYWORDS**

blocking, callback, channel, events, handler, nonblocking.
CrtCloseHdlr

NAME

Tcl_CreateCloseHandler, Tcl_DeleteCloseHandler – arrange for callbacks when channels are closed

SYNOPSIS

#include <tcl.h>

void
Tcl_CreateCloseHandler(channel, proc, clientData)

void
Tcl_DeleteCloseHandler(channel, proc, clientData)

ARGUMENTS

Tcl_Channel channel (in)
    The channel for which to create or delete a close callback.
Tcl_CloseProc *proc (in)
    The procedure to call as the callback.
ClientData clientData (in)
    Arbitrary one-word value to pass to proc.

DESCRIPTION

Tcl_CreateCloseHandler arranges for proc to be called when channel is closed with Tcl_Close or Tcl_UnregisterChannel, or using the Tcl close command. Proc should match the following prototype:

typedef void Tcl_CloseProc(
    ClientData clientData);

The clientData is the same as the value provided in the call to Tcl_CreateCloseHandler.

Tcl_DeleteCloseHandler removes a close callback for channel. The proc and clientData identify which close callback to remove; Tcl_DeleteCloseHandler does nothing if its proc and clientData arguments do not match the proc and clientData for a close handler for channel.

SEE ALSO

close, Tcl_Close, Tcl_UnregisterChannel
KEYWORDS

callback, channel closing

Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
CrtCommand

NAME

Tcl_CreateCommand – implement new commands in C

SYNOPSIS

#include <tcl.h>
Tcl_CmdProc
Tcl_CreateCommand(interp, cmdName, proc, clientData, deleteProc)

ARGUMENTS

Tcl_Interp *interp (in)
   Interpreter in which to create new command.
CONST char *cmdName (in)
   Name of command.
Tcl_CmdProc *proc (in)
   Implementation of new command: proc will be called whenever cmdName is invoked as a command.
ClientData clientData (in)
   Arbitrary one−word value to pass to proc and deleteProc.
Tcl_CmdDeleteProc *deleteProc (in)
   Procedure to call before cmdName is deleted from the interpreter; allows for command−specific cleanup. If NULL, then no procedure is called before the command is deleted.

DESCRIPTION

Tcl_CreateCommand defines a new command in interp and associates it with procedure proc such that whenever cmdName is invoked as a Tcl command (via a call to Tcl_Eval) the Tcl interpreter will call proc to process the command. It differs from Tcl_CreateObjCommand in that a new string−based command is defined; that is, a command procedure is defined that takes an array of argument strings instead of objects. The object−based command procedures registered by Tcl_CreateObjCommand can execute significantly faster than the string−based command procedures defined by Tcl_CreateCommand. This is because they take Tcl objects as arguments and those objects can retain an internal representation that can be manipulated more efficiently. Also, Tcl's interpreter now uses objects internally. In order to invoke a string−based command procedure registered by Tcl_CreateCommand, it must generate and fetch a string representation from each argument object before the call and create a new Tcl object to hold the string result returned by the string−based command procedure. New commands should be defined using Tcl_CreateObjCommand. We support Tcl_CreateCommand for backwards compatibility.

The procedures Tcl_DeleteCommand, Tcl_GetCommandInfo, and Tcl_SetCommandInfo are used in conjunction with Tcl_CreateCommand.
**Tcl_CreateCommand** will delete an existing command *cmdName*, if one is already associated with the interpreter. It returns a token that may be used to refer to the command in subsequent calls to **Tcl_GetCommandName**. If *cmdName* contains any :: namespace qualifiers, then the command is added to the specified namespace; otherwise the command is added to the global namespace. If **Tcl_CreateCommand** is called for an interpreter that is in the process of being deleted, then it does not create a new command and it returns NULL. **Proc** should have arguments and result that match the type **Tcl_CmdProc**:

```c
typedef int Tcl_CmdProc(
    ClientData clientData,
    Tcl_Interp *interp,
    argc
    argv[]);
```

When **proc** is invoked the *clientData* and *interp* parameters will be copies of the *clientData* and *interp* arguments given to **Tcl_CreateCommand**. Typically, *clientData* points to an application−specific data structure that describes what to do when the command procedure is invoked. **Argc** and **argv** describe the arguments to the command. **argc** giving the number of arguments (including the command name) and **argv** giving the values of the arguments as strings. The **argv** array will contain **argc**+1 values; the first **argc** values point to the argument strings, and the last value is NULL. Note that the argument strings should not be modified as they may point to constant strings or may be shared with other parts of the interpreter.

Note that the argument strings are encoded in normalized UTF−8 since version 8.1 of Tcl.

**Proc** must return an integer code that is either **TCL_OK**, **TCL_ERROR**, **TCL_RETURN**, **TCL_BREAK**, or **TCL_CONTINUE**. See the Tcl overview man page for details on what these codes mean. Most normal commands will only return **TCL_OK** or **TCL_ERROR**. In addition, **proc** must set the interpreter result to point to a string value; in the case of a **TCL_OK** return code this gives the result of the command, and in the case of **TCL_ERROR** it gives an error message. The **Tcl_SetResult** procedure provides an easy interface for setting the return value; for complete details on how the the interpreter result field is managed, see the **Tcl_Interp** man page. Before invoking a command procedure, **Tcl_Eval** sets the interpreter result to point to an empty string, so simple commands can return an empty result by doing nothing at all.

The contents of the **argv** array belong to Tcl and are not guaranteed to persist once **proc** returns: **proc** should not modify them, nor should it set the interpreter result to point anywhere within the **argv** values. Call **Tcl_SetResult** with status **TCL_VOLATILE** if you want to return something from the **argv** array.

**DeleteProc** will be invoked when (if) **cmdName** is deleted. This can occur through a call to **Tcl_DeleteCommand** or **Tcl_DeleteInterp**, or by replacing **cmdName** in another call to **Tcl_CreateCommand**. **DeleteProc** is invoked before the command is deleted, and gives the application an opportunity to release any structures associated with the command. **DeleteProc** should have arguments and result that match the type **Tcl_CmdDeleteProc**:

```c
typedef void Tcl_CmdDeleteProc(ClientData clientData);
```

The *clientData* argument will be the same as the *clientData* argument passed to **Tcl_CreateCommand**.
SEE ALSO

Tcl_CreateObjCommand, Tcl_DeleteCommand, Tcl_GetCommandInfo, Tcl_SetCommandInfo,
Tcl_GetCommandName, Tcl_SetObjResult

KEYWORDS

bind, command, create, delete, interpreter, namespace

Copyright ' 1989-1993 The Regents of the University of California.
Copyright ' 1994-1997 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
CrtFileHdlr

NAME

Tcl_CreateFileHandler, Tcl_DeleteFileHandler – associate procedure callbacks with files or devices (Unix only)

SYNOPSIS

#include <tcl.h>
Tcl_CreateFileHandler(fd, mask, proc, clientData)
Tcl_DeleteFileHandler(fd)

ARGUMENTS

int fd (in)
   Unix file descriptor for an open file or device.
int mask (in)
   Conditions under which proc should be called: OR-ed combination of TCL_READABLE, TCL_WRITABLE, and TCL_EXCEPTION. May be set to 0 to temporarily disable a handler.
Tcl_FileProc *proc (in)
   Procedure to invoke whenever the file or device indicated by file meets the conditions specified by mask.
ClientData clientData (in)
   Arbitrary one-word value to pass to proc.

DESCRIPTION

Tcl_CreateFileHandler arranges for proc to be invoked in the future whenever I/O becomes possible on a file or an exceptional condition exists for the file. The file is indicated by fd, and the conditions of interest are indicated by mask. For example, if mask is TCL_READABLE, proc will be called when the file is readable. The callback to proc is made by Tcl_DoOneEvent, so Tcl_CreateFileHandler is only useful in programs that dispatch events through Tcl_DoOneEvent or through Tcl commands such as vwait.

Proc should have arguments and result that match the type Tcl_FileProc:

typedef void Tcl_FileProc(
   ClientData, 
   mask
int

The clientData parameter to proc is a copy of the clientData argument given to Tcl_CreateFileHandler when the callback was created. Typically, clientData points to a data structure containing application-specific information about the file. Mask is an integer mask indicating which of the requested conditions actually exists for the file; it will contain a subset of the bits in the mask argument to Tcl_CreateFileHandler.
There may exist only one handler for a given file at a given time. If \texttt{Tcl\_CreateFileHandler} is called when a handler already exists for \texttt{fd}, then the new callback replaces the information that was previously recorded.

\texttt{Tcl\_DeleteFileHandler} may be called to delete the file handler for \texttt{fd}; if no handler exists for the file given by \texttt{fd} then the procedure has no effect.

The purpose of file handlers is to enable an application to respond to events while waiting for files to become ready for I/O. For this to work correctly, the application may need to use non–blocking I/O operations on the files for which handlers are declared. Otherwise the application may block if it reads or writes too much data; while waiting for the I/O to complete the application won't be able to service other events. Use \texttt{Tcl\_SetChannelOption} with \texttt{–blocking} to set the channel into blocking or nonblocking mode as required.

Note that these interfaces are only supported by the Unix implementation of the Tcl notifier.

\textbf{KEYWORDS}

callback, file, handler
NAME

Tcl_CreateInterp, Tcl_DeleteInterp, Tcl_InterpDeleted – create and delete Tcl command interpreters

SYNOPSIS

ARGUMENTS

DESCRIPTION

INTERPRETERS AND MEMORY MANAGEMENT

Interpreters Passed As Arguments
Interpreter Creation And Deletion
Retrieving An Interpreter From A Data Structure

SEE ALSO

KEYWORDS

NAME

Tcl_CreateInterp, Tcl_DeleteInterp, Tcl_InterpDeleted – create and delete Tcl command interpreters

SYNOPSIS

#include <tcl.h>

Tcl_Interp *
Tcl_CreateInterp()
Tcl_DeleteInterp(interp)
int
Tcl_InterpDeleted(interp)

ARGUMENTS

Tcl_Interp *interp (in)
Token for interpreter to be destroyed.

DESCRIPTION

Tcl_CreateInterp creates a new interpreter structure and returns a token for it. The token is required in calls to most other Tcl procedures, such as Tcl_CreateCommand, Tcl_Eval, and Tcl_DeleteInterp. Clients are only allowed to access a few of the fields of Tcl_Interp structures; see the Tcl_Interp and Tcl_CreateCommand man pages for details. The new interpreter is initialized with the built–in Tcl commands and with the variables documented in tclvars(n). To bind in additional commands, call Tcl_CreateCommand.
Tcl_DeleteInterp marks an interpreter as deleted; the interpreter will eventually be deleted when all calls to Tcl_Preserve for it have been matched by calls to Tcl_Release. At that time, all of the resources associated with it, including variables, procedures, and application-specific command bindings, will be deleted. After Tcl_DeleteInterp returns any attempt to use Tcl_Eval on the interpreter will fail and return TCL_ERROR. After the call to Tcl_DeleteInterp it is safe to examine the interpreter's result, query or set the values of variables, define, undefine or retrieve procedures, and examine the runtime evaluation stack. See below, in the section INTERPRETERS AND MEMORY MANAGEMENT for details.

Tcl_InterpDeleted returns nonzero if Tcl_DeleteInterp was called with interp as its argument; this indicates that the interpreter will eventually be deleted, when the last call to Tcl_Preserve for it is matched by a call to Tcl_Release. If nonzero is returned, further calls to Tcl_Eval in this interpreter will return TCL_ERROR.

Tcl_InterpDeleted is useful in deletion callbacks to distinguish between when only the memory the callback is responsible for is being deleted and when the whole interpreter is being deleted. In the former case the callback may recreate the data being deleted, but this would lead to an infinite loop if the interpreter were being deleted.

INTERPRETERS AND MEMORY MANAGEMENT

Tcl_DeleteInterp can be called at any time on an interpreter that may be used by nested evaluations and C code in various extensions. Tcl implements a simple mechanism that allows callers to use interpreters without worrying about the interpreter being deleted in a nested call, and without requiring special code to protect the interpreter, in most cases. This mechanism ensures that nested uses of an interpreter can safely continue using it even after Tcl_DeleteInterp is called.

The mechanism relies on matching up calls to Tcl_Preserve with calls to Tcl_Release. If Tcl_DeleteInterp has been called, only when the last call to Tcl_Preserve is matched by a call to Tcl_Release, will the interpreter be freed. See the manual entry for Tcl_Preserve for a description of these functions.

The rules for when the user of an interpreter must call Tcl_Preserve and Tcl_Release are simple:

Interpreters Passed As Arguments
Functions that are passed an interpreter as an argument can safely use the interpreter without any special protection. Thus, when you write an extension consisting of new Tcl commands, no special code is needed to protect interpreters received as arguments. This covers the majority of all uses.

Interpreter Creation And Deletion
When a new interpreter is created and used in a call to Tcl_Eval, Tcl_VarEval, Tcl_GlobalEval, Tcl_SetVar, or Tcl_GetVar, a pair of calls to Tcl_Preserve and Tcl_Release should be wrapped around all uses of the interpreter. Remember that it is unsafe to use the interpreter once Tcl_Release has been called. To ensure that the interpreter is properly deleted when it is no longer needed, call Tcl_InterpDeleted to test if some other code already called Tcl_DeleteInterp; if not, call Tcl_DeleteInterp before calling Tcl_Release in your own code.

Retrieving An Interpreter From A Data Structure
When an interpreter is retrieved from a data structure (e.g. the client data of a callback) for use in Tcl_Eval, Tcl_VarEval, Tcl_GlobalEval, Tcl_SetVar, or Tcl_GetVar, a pair of calls to
**Tcl_Preserve** and **Tcl_Release** should be wrapped around all uses of the interpreter; it is unsafe to reuse the interpreter once **Tcl_Release** has been called. If an interpreter is stored inside a callback data structure, an appropriate deletion cleanup mechanism should be set up by the code that creates the data structure so that the interpreter is removed from the data structure (e.g. by setting the field to NULL) when the interpreter is deleted. Otherwise, you may be using an interpreter that has been freed and whose memory may already have been reused.

All uses of interpreters in Tcl and Tk have already been protected. Extension writers should ensure that their code also properly protects any additional interpreters used, as described above.

**SEE ALSO**

**Tcl_Preserve, Tcl_Release**

**KEYWORDS**

command, create, delete, interpreter

---

Copyright © 1989-1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
Copyright © 1995-1997 Roger E. Critchlow Jr.
NAME

Tcl_CreateMathFunc, Tcl_GetMathFuncInfo, Tcl_ListMathFuncs – Define, query and enumerate math
functions for expressions

SYNOPSIS

#include <tcl.h>

void
Tcl_CreateMathFunc(interp, name, numArgs, argTypes, proc, clientData)

int
Tcl_GetMathFuncInfo(interp, name, numArgsPtr, argTypesPtr, procPtr, clientDataPtr)

Tcl_Obj *
Tcl_ListMathFuncs(interp, pattern)

ARGUMENTS

Tcl_Interp *interp (in)

Interpreter in which new function will be defined.

CONST char *name (in)

Name for new function.

int numArgs (in)

Number of arguments to new function; also gives size of argTypes array.

Tcl_ValueType *argTypes (in)

Points to an array giving the permissible types for each argument to function.

Tcl_MathProc *proc (in)

Procedure that implements the function.

ClientData clientData (in)

Arbitrary one-word value to pass to proc when it is invoked.

int *numArgsPtr (out)

Points to a variable that will be set to contain the number of arguments to the function.

Tcl_ValueType **argTypesPtr (out)

Points to a variable that will be set to contain a pointer to an array giving the permissible types for each argument to the function which will need to be freed up using Tcl_Free.

Tcl_MathProc **procPtr (out)

Points to a variable that will be set to contain a pointer to the implementation code for the function (or NULL if the function is implemented directly in bytecode.)

ClientData *clientDataPtr (out)

Points to a variable that will be set to contain the clientData argument passed to Tcl_CreateMathFunc when the function was created if the function is not implemented directly in bytecode.

CONST char *pattern (in)

Pattern to match against function names so as to filter them (by passing to Tcl_StringMatch), or
DESCRIPTION

Tcl allows a number of mathematical functions to be used in expressions, such as \texttt{sin}, \texttt{cos}, and \texttt{hypot}. \texttt{Tcl\_CreateMathFunc} allows applications to add additional functions to those already provided by Tcl or to replace existing functions. \textit{Name} is the name of the function as it will appear in expressions. If \textit{name} doesn't already exist as a function then a new function is created. If it does exist, then the existing function is replaced. \textit{NumArgs} and \textit{argTypes} describe the arguments to the function. Each entry in the \textit{argTypes} array must be one of \texttt{TCL\_INT}, \texttt{TCL\_DOUBLE}, \texttt{TCL\_WIDE\_INT}, or \texttt{TCL\_EITHER} to indicate whether the corresponding argument must be an integer, a double-precision floating value, a wide (64-bit) integer, or any, respectively.

Whenever the function is invoked in an expression Tcl will invoke \textit{proc}. \textit{Proc} should have arguments and result that match the type \texttt{Tcl\_MathProc}:

```c
typedef int Tcl_MathProc(
    ClientData clientData,
    Tcl_Interp * interp,
    Tcl_Value * args,
    Tcl_Value * resultPtr);
```

When \textit{proc} is invoked the \textit{clientData} and \textit{interp} arguments will be the same as those passed to \texttt{Tcl\_CreateMathFunc}. \textit{Args} will point to an array of \textit{numArgs} \texttt{Tcl\_Value} structures, which describe the actual arguments to the function:

```c
typedef struct Tcl_Value {
    Tcl_ValueType type;
    long intValue;
    double doubleValue;
    Tcl_WideInt wideValue;
} Tcl_Value;
```

The \textit{type} field indicates the type of the argument and is one of \texttt{TCL\_INT}, \texttt{TCL\_DOUBLE} or \texttt{TCL\_WIDE\_INT}. It will match the \textit{argTypes} value specified for the function unless the \textit{argTypes} value was \texttt{TCL\_EITHER}. Tcl converts the argument supplied in the expression to the type requested in \textit{argTypes}, if that is necessary. Depending on the value of the \textit{type} field, the \textit{intValue}, \textit{doubleValue} or \textit{wideValue} field will contain the actual value of the argument.

\textit{Proc} should compute its result and store it either as an integer in \textit{resultPtr->intValue} or as a floating value in \textit{resultPtr->doubleValue}. It should set also \textit{resultPtr->type} to one of \texttt{TCL\_INT}, \texttt{TCL\_DOUBLE} or \texttt{TCL\_WIDE\_INT} to indicate which value was set. Under normal circumstances \textit{proc} should return \texttt{TCL\_OK}. If an error occurs while executing the function, \textit{proc} should return \texttt{TCL\_ERROR} and leave an error message in the interpreter's result.

\texttt{Tcl\_GetMathFuncInfo} retrieves the values associated with function \textit{name} that were passed to a preceding \texttt{Tcl\_CreateMathFunc} call. Normally, the return code is \texttt{TCL\_OK} but if the named function does not exist,
TCL_ERROR is returned and an error message is placed in the interpreter's result.

If an error did not occur, the array reference placed in the variable pointed to by argTypesPtr is newly allocated, and should be released by passing it to Tcl_Free. Some functions (the standard set implemented in the core) are implemented directly at the bytecode level; attempting to retrieve values for them causes a NULL to be stored in the variable pointed to by procPtr and the variable pointed to by clientDataPtr will not be modified.

Tcl_ListMathFuncs returns a Tcl object containing a list of all the math functions defined in the interpreter whose name matches pattern. In the case of an error, NULL is returned and an error message is left in the interpreter result, and otherwise the returned object will have a reference count of zero.

KEYWORDS

expression, mathematical function

SEE ALSO

expr_info, Tcl_Free, Tcl_NewListObj

Copyright ' 1989−1993 The Regents of the University of California.
Copyright ' 1994−1996 Sun Microsystems, Inc.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
CrtObjCmd

NAME

Tcl_CreateObjCommand, Tcl_DeleteCommand, Tcl_DeleteCommandFromToken, Tcl_GetCommandInfo, Tcl_GetCommandInfoFromToken, Tcl_SetCommandInfo, Tcl_SetCommandInfoFromToken, Tcl_GetCommandName, Tcl_GetCommandFullName, Tcl_GetCommandFromObj – implement new commands in C

SYNOPSIS

#include <tcl.h>
Tcl_Command
Tcl_CreateObjCommand(interp, cmdName, proc, clientData, deleteProc)
int
Tcl_DeleteCommand(interp, cmdName)
int
Tcl_DeleteCommandFromToken(interp, token)
int
Tcl_GetCommandInfo(interp, cmdName, infoPtr)
int
Tcl_SetCommandInfo(interp, cmdName, infoPtr)
int
Tcl_GetCommandInfoFromToken(token, infoPtr)
int
Tcl_SetCommandInfoFromToken(token, infoPtr)
CONST char *
Tcl_GetCommandName(interp, token)
void
Tcl_GetCommandFullName(interp, token, objPtr)
Tcl_Command
Tcl_GetCommandFromObj(interp, objPtr)

ARGUMENTS

Tcl_Interp *interp (in)
   Interpreter in which to create a new command or that contains a command.
char *cmdName (in)
   Name of command.
Tcl_ObjCmdProc *proc (in)
   Implementation of the new command: proc will be called whenever cmdName is invoked as a command.
ClientData clientData (in)
   Arbitrary one-word value to pass to proc and deleteProc.
**Tcl_CmdDeleteProc */**deleteProc*(in)**

Procedure to call before *cmdName* is deleted from the interpreter; allows for command-specific cleanup. If NULL, then no procedure is called before the command is deleted.

**Tcl_Command*/**token*(in)**

Token for command, returned by previous call to Tcl_CreateObjCommand. The command must not have been deleted.

**Tcl_CmdInfo */**infoPtr*(in/out)**

Pointer to structure containing various information about a Tcl command.

**Tcl_Obj */**objPtr*(in)**

Object containing the name of a Tcl command.

---

**DESCRIPTION**

**Tcl_CreateObjCommand** defines a new command in *interp* and associates it with procedure *proc* such that whenever *name* is invoked as a Tcl command (e.g., via a call to Tcl_EvalObjEx) the Tcl interpreter will call *proc* to process the command.

**Tcl_CreateObjCommand** deletes any existing command *name* already associated with the interpreter (however see below for an exception where the existing command is not deleted). It returns a token that may be used to refer to the command in subsequent calls to Tcl_GetCommandName. If *name* contains any :: namespace qualifiers, then the command is added to the specified namespace; otherwise the command is added to the global namespace. If Tcl_CreateObjCommand is called for an interpreter that is in the process of being deleted, then it does not create a new command and it returns NULL. *proc* should have arguments and result that match the type Tcl_ObjCmdProc:

```c
typedef int Tcl_ObjCmdProc(
    ClientData clientData,
    Tcl_Interp *interp,
    int objc,
    Tcl_Obj *CONST objv[]);
```

When *proc* is invoked, the *clientData* and *interp* parameters will be copies of the *clientData* and *interp* arguments given to Tcl_CreateObjCommand. Typically, *clientData* points to an application–specific data structure that describes what to do when the command procedure is invoked. *Objc* and *objv* describe the arguments to the command, *objc* giving the number of argument objects (including the command name) and *objv* giving the values of the arguments. The *objv* array will contain *objc* values, pointing to the argument objects. Unlike argv[argv] used in a string–based command procedure, *objv*[objc] will not contain NULL.

Additionally, when *proc* is invoked, it must not modify the contents of the *objv* array by assigning new pointer values to any element of the array (for example, *objv*[2] = NULL) because this will cause memory to be lost and the runtime stack to be corrupted. The **CONST** in the declaration of *objv* will cause ANSI–compliant compilers to report any such attempted assignment as an error. However, it is acceptable to modify the internal representation of any individual object argument. For instance, the user may call Tcl_GetIntFromObj on *objv*[2] to obtain the integer representation of that object; that call may change the type of the object that *objv*[2] points at, but will not change where *objv*[2] points.
proc must return an integer code that is either TCL_OK, TCL_ERROR, TCL_RETURN, TCL_BREAK, or TCL_CONTINUE. See the Tcl overview man page for details on what these codes mean. Most normal commands will only return TCL_OK or TCL_ERROR. In addition, if proc needs to return a non-empty result, it can call Tcl_SetObjResult to set the interpreter's result. In the case of a TCL_OK return code this gives the result of the command, and in the case of TCL_ERROR this gives an error message. Before invoking a command procedure, Tcl_EvalObjEx sets interpreter's result to point to an object representing an empty string, so simple commands can return an empty result by doing nothing at all.

The contents of the objv array belong to Tcl and are not guaranteed to persist once proc returns: proc should not modify them. Call Tcl_SetObjResult if you want to return something from the objv array.

Ordinarily, Tcl_CreateObjCommand deletes any existing command name already associated with the interpreter. However, if the existing command was created by a previous call to Tcl_CreateCommand, Tcl_CreateObjCommand does not delete the command but instead arranges for the Tcl interpreter to call the Tcl_ObjCmdProc proc in the future. The old string-based Tcl_CmdProc associated with the command is retained and its address can be obtained by subsequent Tcl_GetCommandInfo calls. This is done for backwards compatibility.

DeleteProc will be invoked when (if) name is deleted. This can occur through a call to Tcl_DeleteCommand, Tcl_DeleteCommandFromToken, or Tcl_DeleteInterp, or by replacing name in another call to Tcl_CreateObjCommand. DeleteProc is invoked before the command is deleted, and gives the application an opportunity to release any structures associated with the command. DeleteProc should have arguments and result that match the type Tcl_CmdDeleteProc:

typedef void Tcl_CmdDeleteProc(ClientData clientData);

The clientData argument will be the same as the clientData argument passed to Tcl_CreateObjCommand.

Tcl_DeleteCommand deletes a command from a command interpreter. Once the call completes, attempts to invoke cmdName in interp will result in errors. If cmdName isn't bound as a command in interp then Tcl_DeleteCommand does nothing and returns −1; otherwise it returns 0. There are no restrictions on cmdName: it may refer to a built-in command, an application-specific command, or a Tcl procedure. If name contains any :: namespace qualifiers, the command is deleted from the specified namespace.

Given a token returned by Tcl_CreateObjCommand, Tcl_DeleteCommandFromToken deletes the command from a command interpreter. It will delete a command even if that command has been renamed. Once the call completes, attempts to invoke the command in interp will result in errors. If the command corresponding to token has already been deleted from interp then Tcl_DeleteCommand does nothing and returns −1; otherwise it returns 0.

Tcl_GetCommandInfo checks to see whether its cmdName argument exists as a command in interp. cmdName may include :: namespace qualifiers to identify a command in a particular namespace. If the command is not found, then it returns 0. Otherwise it places information about the command in the Tcl_CmdInfo structure pointed to by infoPtr and returns 1. A Tcl_CmdInfo structure has the following fields:
typedef struct Tcl_CmdInfo {
    int isNativeObjectProc;
    Tcl_CmdProc *objProc;
    ClientData objClientData;
    Tcl_CmdProc *proc;
    ClientData clientData;
    Tcl_CmdDeleteProc *deleteProc;
    ClientData deleteData;
    Tcl_Namespace *namespacePtr;
} Tcl_CmdInfo;

The isNativeObjectProc field has the value 1 if Tcl_CreateObjCommand was called to register the command; it is 0 if only Tcl_CreateCommand was called. It allows a program to determine whether it is faster to call objProc or proc: objProc is normally faster if isNativeObjectProc has the value 1. The fields objProc and objClientData have the same meaning as the proc and clientData arguments to Tcl_CreateObjCommand; they hold information about the object-based command procedure that the Tcl interpreter calls to implement the command. The fields proc and clientData hold information about the string-based command procedure that implements the command. If Tcl_CreateCommand was called for this command, this is the procedure passed to it; otherwise, this is a compatibility procedure registered by Tcl_CreateObjCommand that simply calls the command's object-based procedure after converting its string arguments to Tcl objects. The field deleteData is the ClientData value to pass to deleteProc; it is normally the same as clientData but may be set independently using the Tcl_SetCommandInfo procedure. The field namespacePtr holds a pointer to the Tcl_Namespace that contains the command.

Tcl_GetCommandInfoFromToken is identical to Tcl_GetCommandInfo except that it uses a command token returned from Tcl_CreateObjCommand in place of the command name. If the token parameter is NULL, it returns 0; otherwise, it returns 1 and fills in the structure designated by infoPtr.

Tcl_SetCommandInfo is used to modify the procedures and ClientData values associated with a command. Its cmdName argument is the name of a command in interp. cmdName may include :: namespace qualifiers to identify a command in a particular namespace. If this command does not exist then Tcl_SetCommandInfo returns 0. Otherwise, it copies the information from *infoPtr to Tcl's internal structure for the command and returns 1.

Tcl_SetCommandInfoFromToken is identical to Tcl_SetCommandInfo except that it takes a command token as returned by Tcl_CreateObjCommand instead of the command name. If the token parameter is NULL, it returns 0. Otherwise, it copies the information from *infoPtr to Tcl's internal structure for the command and returns 1.

Note that Tcl_SetCommandInfo and Tcl_SetCommandInfoFromToken both allow the ClientData for a command's deletion procedure to be given a different value than the ClientData for its command procedure.

Note that neither Tcl_SetCommandInfo nor Tcl_SetCommandInfoFromToken will change a command's namespace. Use Tcl_Eval to call the rename command to do that.

Tcl_GetCommandName provides a mechanism for tracking commands that have been renamed. Given a token returned by Tcl_CreateObjCommand when the command was created, Tcl_GetCommandName
returns the string name of the command. If the command has been renamed since it was created, then
\textbf{Tcl\_GetCommandName} returns the current name. This name does not include any :: namespace qualifiers.
The command corresponding to \textit{token} must not have been deleted. The string returned by
\textbf{Tcl\_GetCommandName} is in dynamic memory owned by Tcl and is only guaranteed to retain its value as long as the command isn't deleted or renamed; callers should copy the string if they need to keep it for a long time.

\textbf{Tcl\_GetCommandFullName} produces the fully–qualified name of a command from a command token. The name, including all namespace prefixes, is appended to the object specified by \textit{objPtr}.

\textbf{Tcl\_GetCommandFromObj} returns a token for the command specified by the name in a \textbf{Tcl\_Obj}. The command name is resolved relative to the current namespace. Returns NULL if the command is not found.

\textbf{SEE ALSO}

\textbf{Tcl\_CreateCommand, Tcl\_ResetResult, Tcl\_SetObjResult}

\textbf{KEYWORDS}

\textit{bind, command, create, delete, namespace, object}

\texttt{Copyright} ' 1996-1997 Sun Microsystems, Inc.
\texttt{Copyright} ' 1995-1997 Roger E. Critchlow Jr.
CrtSlave

NAME

Tcl_IsSafe, Tcl_MakeSafe, Tcl_CreateSlave, Tcl_GetSlave, Tcl_GetMaster, Tcl_GetInterpPath,
Tcl_CreateAlias, Tcl_CreateAliasObj, Tcl_GetAlias, Tcl_GetAliasObj, Tcl_ExposeCommand,
Tcl_HideCommand – manage multiple Tcl interpreters, aliases and hidden commands.

SYNOPSIS

#include <tcl.h>
#include <tcl.h>
int
#include <tcl.h>
Tcl_IsSafe(interp)
#include <tcl.h>
Tcl_MakeSafe(interp)
#include <tcl.h>
Tcl_CreateSlave(interp, slaveName, isSafe)
#include <tcl.h>
Tcl_GetSlave(interp, slaveName)
#include <tcl.h>
Tcl_GetMaster(interp)
#include <tcl.h>
Tcl_GetInterpPath(askingInterp, slaveInterp)
#include <tcl.h>
Tcl_CreateAlias(slaveInterp, slaveCmd, targetInterp, targetCmd, argc, argv)
#include <tcl.h>
Tcl_CreateAliasObj(slaveInterp, slaveCmd, targetInterp, targetCmd, objc, objv)
#include <tcl.h>
Tcl_GetAlias(interp, slaveCmd, targetInterpPtr, targetCmdPtr, argcPtr, argvPtr)
#include <tcl.h>
Tcl_GetAliasObj(interp, slaveCmd, targetInterpPtr, targetCmdPtr, objcPtr, objvPtr)
#include <tcl.h>
Tcl_ExposeCommand(interp, hiddenCmdName, cmdName)
#include <tcl.h>
Tcl_HideCommand(interp, cmdName, hiddenCmdName)

ARGUMENTS

Tcl_Interp *interp (in)
    Interpreter in which to execute the specified command.

CONST char *slaveName (in)
    Name of slave interpreter to create or manipulate.

int isSafe (in)
    If non-zero, a ``safe'' slave that is suitable for running untrusted code is created, otherwise a trusted
slave is created.

*Tcl_Interp*

**slaveInterp** *(in)*

Interpreter to use for creating the source command for an alias (see below).

*CONST char*

**slaveCmd** *(in)*

Name of source command for alias.

*Tcl_Interp*

**targetInterp** *(in)*

Interpreter that contains the target command for an alias.

*CONST char*

**targetCmd** *(in)*

Name of target command for alias in *targetInterp*.

int

**argc** *(in)*

Count of additional arguments to pass to the alias command.

*CONST char* *CONST*

**argv** *(in)*

Vector of strings, the additional arguments to pass to the alias command. This storage is owned by the caller.

int

**objc** *(in)*

Count of additional object arguments to pass to the alias object command.

*Tcl_Object*

**objv** *(in)*

Vector of Tcl_Obj structures, the additional object arguments to pass to the alias object command. This storage is owned by the caller.

*Tcl_Interp*

**targetInterpPtr** *(in)*

Pointer to location to store the address of the interpreter where a target command is defined for an alias.

*CONST char* *CONST*

**targetCmdPtr** *(out)*

Pointer to location to store the address of the name of the target command for an alias.

int

**argcPtr** *(out)*

Pointer to location to store count of additional arguments to be passed to the alias. The location is in storage owned by the caller.

*CONST char* *CONST*

**argvPtr** *(out)*

Pointer to location to store a vector of strings, the additional arguments to pass to an alias. The location is in storage owned by the caller, the vector of strings is owned by the called function.

int

**objcPtr** *(out)*

Pointer to location to store count of additional object arguments to be passed to the alias. The location is in storage owned by the caller.

*Tcl_Obj***

**objvPtr** *(out)*

Pointer to location to store a vector of Tcl_Obj structures, the additional arguments to pass to an object alias command. The location is in storage owned by the caller, the vector of Tcl_Obj structures is owned by the called function.

*CONST char*

**cmdName** *(in)*

Name of an exposed command to hide or create.

*CONST char*

**hiddenCmdName** *(in)*

Name under which a hidden command is stored and with which it can be exposed or invoked.
DESCRIPTION

These procedures are intended for access to the multiple interpreter facility from inside C programs. They enable managing multiple interpreters in a hierarchical relationship, and the management of aliases, commands that when invoked in one interpreter execute a command in another interpreter. The return value for those procedures that return an int is either TCL_OK or TCL_ERROR. If TCL_ERROR is returned then the result field of the interpreter contains an error message.

Tcl_CreateSlave creates a new interpreter as a slave of interp. It also creates a slave command named slaveName in interp which allows interp to manipulate the new slave. If isSafe is zero, the command creates a trusted slave in which Tcl code has access to all the Tcl commands. If it is 1, the command creates a ```safe``` slave in which Tcl code has access only to set of Tcl commands defined as ```Safe Tcl```; see the manual entry for the Tcl interp command for details. If the creation of the new slave interpreter failed, NULL is returned.

Tcl_IsSafe returns 1 if interp is ```safe``` (was created with the TCL_SAFE_INTERPRETER flag specified), 0 otherwise.

Tcl_MakeSafe marks interp as ```safe```, so that future calls to Tcl_IsSafe will return 1. It also removes all known potentially−unsafe core functionality (both commands and variables) from interp. However, it cannot know what parts of an extension or application are safe and does not make any attempt to remove those parts, so safety is not guaranteed after calling Tcl_MakeSafe. Callers will want to take care with their use of Tcl_MakeSafe to avoid false claims of safety. For many situations, Tcl_CreateSlave may be a better choice, since it creates interpreters in a known−safe state.

Tcl_GetSlave returns a pointer to a slave interpreter of interp. The slave interpreter is identified by slaveName. If no such slave interpreter exists, NULL is returned.

Tcl_GetMaster returns a pointer to the master interpreter of interp. If interp has no master (it is a top−level interpreter) then NULL is returned.

Tcl_GetInterpPath sets the result field in askingInterp to the relative path between askingInterp and slaveInterp; slaveInterp must be a slave of askingInterp. If the computation of the relative path succeeds, TCL_OK is returned, else TCL_ERROR is returned and the result field in askingInterp contains the error message.

Tcl_CreateAlias creates an object command named slaveCmd in slaveInterp that when invoked, will cause the command targetCmd to be invoked in targetInterp. The arguments specified by the strings contained in argv are always prepended to any arguments supplied in the invocation of slaveCmd and passed to targetCmd. This operation returns TCL_OK if it succeeds, or TCL_ERROR if it fails; in that case, an error message is left in the object result of slaveInterp. Note that there are no restrictions on the ancestry relationship (as created by Tcl_CreateSlave) between slaveInterp and targetInterp. Any two interpreters can be used, without any restrictions on how they are related.

Tcl_CreateAliasObj is similar to Tcl_CreateAlias except that it takes a vector of objects to pass as additional arguments instead of a vector of strings.
Tcl_GetAlias returns information about an alias aliasName in interp. Any of the result fields can be NULL, in which case the corresponding datum is not returned. If a result field is non–NULL, the address indicated is set to the corresponding datum. For example, if targetNamePtr is non–NULL it is set to a pointer to the string containing the name of the target command.

Tcl_GetAliasObj is similar to Tcl_GetAlias except that it returns a pointer to a vector of Tcl_Obj structures instead of a vector of strings.

Tcl_ExposeCommand moves the command named hiddenCmdName from the set of hidden commands to the set of exposed commands, putting it under the name cmdName.HiddenCmdName must be the name of an existing hidden command, or the operation will return TCL_ERROR and leave an error message in the result field in interp. If an exposed command named cmdName already exists, the operation returns TCL_ERROR and leaves an error message in the object result of interp. If the operation succeeds, it returns TCL_OK. After executing this command, attempts to use cmdName in a call to Tcl_Eval or with the Tcl eval command will again succeed.

Tcl_HideCommand moves the command named cmdName from the set of exposed commands to the set of hidden commands, under the name hiddenCmdName. CmdName must be the name of an existing exposed command, or the operation will return TCL_ERROR and leave an error message in the object result of interp. Currently both cmdName and hiddenCmdName must not contain namespace qualifiers, or the operation will return TCL_ERROR and leave an error message in the object result of interp. The CmdName will be looked up in the global namespace, and not relative to the current namespace, even if the current namespace is not the global one. If a hidden command whose name is hiddenCmdName already exists, the operation also returns TCL_ERROR and the result field in interp contains an error message. If the operation succeeds, it returns TCL_OK. After executing this command, attempts to use cmdName in a call to Tcl_Eval or with the Tcl eval command will fail.

For a description of the Tcl interface to multiple interpreters, see interp(n).

SEE ALSO

interp

KEYWORDS

alias, command, exposed commands, hidden commands, interpreter, invoke, master, slave

Copyright ' 1995−1996 Sun Microsystems, Inc.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
CrtTimerHdlr

NAME

Tcl_CreateTimerHandler, Tcl_DeleteTimerHandler – call a procedure at a given time

SYNOPSIS

#include <tcl.h>
Tcl_TimerToken
Tcl_CreateTimerHandler(milliseconds, proc, clientData)
Tcl_DeleteTimerHandler(token)

ARGUMENTS

int milliseconds (in)
    How many milliseconds to wait before invoking proc.

Tcl_TimerProc *proc (in)
    Procedure to invoke after milliseconds have elapsed.

ClientData clientData (in)
    Arbitrary one-word value to pass to proc.

Tcl_TimerToken token (in)
    Token for previously-created timer handler (the return value from some previous call to Tcl_CreateTimerHandler).

DESCRIPTION

Tcl_CreateTimerHandler arranges for proc to be invoked at a time milliseconds milliseconds in the future. The callback to proc will be made by Tcl_DoOneEvent, so Tcl_CreateTimerHandler is only useful in programs that dispatch events through Tcl_DoOneEvent or through Tcl commands such as vwait. The call to proc may not be made at the exact time given by milliseconds: it will be made at the next opportunity after that time. For example, if Tcl_DoOneEvent isn't called until long after the time has elapsed, or if there are other pending events to process before the call to proc, then the call to proc will be delayed.

Proc should have arguments and return value that match the type Tcl_TimerProc:

typedef void Tcl_TimerProc(ClientData clientData);

The clientData parameter to proc is a copy of the clientData argument given to Tcl_CreateTimerHandler when the callback was created. Typically, clientData points to a data structure containing application-specific information about what to do in proc.

Tcl_DeleteTimerHandler may be called to delete a previously-created timer handler. It deletes the handler indicated by token so that no call to proc will be made; if that handler no longer exists (e.g. because the time
period has already elapsed and proc has been invoked then Tcl_DeleteTimerHandler does nothing. The tokens returned by Tcl_CreateTimerHandler never have a value of NULL, so if NULL is passed to Tcl_DeleteTimerHandler then the procedure does nothing.

KEYWORDS

callback, clock, handler, timer

Copyright ' 1990 The Regents of the University of California.
Copyright ' 1994−1996 Sun Microsystems, Inc.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
NAME

Tcl_CreateTrace, Tcl_CreateObjTrace, Tcl_DeleteTrace – arrange for command execution to be traced

SYNOPSIS

```
#include <tcl.h>
Tcl_Trace
Tcl_CreateTrace(interp, level, proc, clientData)
Tcl_Trace
Tcl_CreateObjTrace(interp, level, flags, objProc, clientData, deleteProc)
Tcl_Trace
Tcl_DeleteTrace(interp, trace)
```

ARGUMENTS

*Tcl_Interp* interp (in)

Interpreter containing command to be traced or untraced.

int level (in)

Only commands at or below this nesting level will be traced unless 0 is specified. 1 means top–level commands only, 2 means top–level commands or those that are invoked as immediate consequences of executing top–level commands (procedure bodies, bracketed commands, etc.) and so on. A value of 0 means that commands at any level are traced.

int flags (in)

Flags governing the trace execution. See below for details.

*Tcl_CmdObjTraceProc* objProc (in)

Procedure to call for each command that's executed. See below for details of the calling sequence.

*Tcl_CmdTraceProc* proc (in)

Procedure to call for each command that's executed. See below for details on the calling sequence.

ClientData clientData (in)

Arbitrary one–word value to pass to objProc or proc.

*Tcl_CmdObjTraceDeleteProc* deleteProc ()

Procedure to call when the trace is deleted. See below for details of the calling sequence. A NULL pointer is permissible and results in no callback when the trace is deleted.

*Tcl_Trace* trace (in)

Token for trace to be removed (return value from previous call to Tcl_CreateTrace).

DESCRIPTION

*Tcl_CreateObjTrace* arranges for command tracing. After it is called, objProc will be invoked before the Tcl interpreter calls any command procedure when evaluating commands in interp. The return value from Tcl_CreateObjTrace is a token for the trace, which may be passed to Tcl_DeleteTrace to remove the trace. There may be many traces in effect simultaneously for the same interpreter.
**objProc** should have arguments and result that match the type, **Tcl_CmdObjTraceProc**:

```c
typedef int Tcl_CmdObjTraceProc(
    ClientData clientData,
    Tcl_Interp* interp,
    int level,
    CONST char* command,
    Tcl_Command commandToken,
    int objc,
    Tcl_Obj *CONST objv[]);
```

The *clientData* and *interp* parameters are copies of the corresponding arguments given to **Tcl_CreateTrace**. *ClientData* typically points to an application–specific data structure that describes what to do when *objProc* is invoked. The *level* parameter gives the nesting level of the command (1 for top–level commands passed to **Tcl_CmdObjTraceProc** by the application, 2 for the next–level commands passed to **Tcl_CmdObjTraceProc** as part of parsing or interpreting level–1 commands, and so on). The *command* parameter points to a string containing the text of the command, before any argument substitution. The *commandToken* parameter is a Tcl command token that identifies the command to be invoked. The token may be passed to **Tcl_GetCommandName**, **Tcl_GetCommandTokenInfo**, or **Tcl_SetCommandTokenInfo** to manipulate the definition of the command. The *objc* and *objv* parameters designate the final parameter count and parameter vector that will be passed to the command, and have had all substitutions performed.

The *objProc* callback is expected to return a standard Tcl status return code. If this code is **TCL_OK** (the normal case), then the Tcl interpreter will invoke the command. Any other return code is treated as if the command returned that status, and the command is **not** invoked.

The *objProc* callback must not modify *objv* in any way. It is, however, permissible to change the command by calling **Tcl_SetCommandTokenInfo** prior to returning. Any such change takes effect immediately, and the command is invoked with the new information.

Tracing will only occur for commands at nesting level less than or equal to the *level* parameter (i.e. the *level* parameter to *objProc* will always be less than or equal to the *level* parameter to **Tcl_CreateTrace**).

Tracing has a significant effect on runtime performance because it causes the bytecode compiler to refrain from generating in–line code for Tcl commands such as **if** and **while** in order that they may be traced. If traces for the built–in commands are not required, the *flags* parameter may be set to the constant value **TCL_ALLOW_INLINE_COMPILATION**. In this case, traces on built–in commands may or may not result in trace callbacks, depending on the state of the interpreter, but run–time performance will be improved significantly. (This functionality is desirable, for example, when using **Tcl_CreateObjTrace** to implement an execution time profiler.)

Calls to *objProc* will be made by the Tcl parser immediately before it calls the command procedure for the command (**cmdProc**). This occurs after argument parsing and substitution, so tracing for substituted commands occurs before tracing of the commands containing the substitutions. If there is a syntax error in a command, or if there is no command procedure associated with a command name, then no tracing will occur for that command. If a string passed to **Tcl_CmdObjTraceProc** contains multiple commands (bracketed, or on different lines) then multiple calls to *objProc* will occur, one for each command.
**Tcl_DeleteTrace** removes a trace, so that no future calls will be made to the procedure associated with the trace. After **Tcl_DeleteTrace** returns, the caller should never again use the *trace* token.

When **Tcl_DeleteTrace** is called, the interpreter invokes the *deleteProc* that was passed as a parameter to **Tcl_CreateObjTrace**. The *deleteProc* must match the type, **Tcl_CmdObjTraceDeleteProc**:

```c
typedef void Tcl_CmdObjTraceDeleteProc(
    ClientData clientData
);
```

The *clientData* parameter will be the same as the *clientData* parameter that was originally passed to **Tcl_CreateObjTrace**.

**Tcl_CreateTrace** is an alternative interface for command tracing, *not recommended for new applications*. It is provided for backward compatibility with code that was developed for older versions of the Tcl interpreter. It is similar to **Tcl_CreateObjTrace**, except that its *proc* parameter should have arguments and result that match the type **Tcl_CmdTraceProc**:

```c
typedef void Tcl_CmdTraceProc(
    ClientData clientData,
    Tcl_Interp *interp,
    levelint level,
    char *command;
    Tcl_CmdProc *cmdProc,
    ClientData cmdClientData,
    argcint argc,
    argvconst char *argv[]);
```

The parameters to the *proc* callback are similar to those of the *objProc* callback above. The *commandToken* is replaced with *cmdProc*, a pointer to the (string-based) command procedure that will be invoked; and *cmdClientData*, the client data that will be passed to the procedure. The *objc* parameter is replaced with an *argv* parameter, that gives the arguments to the command as character strings. *Proc* must not modify the *command* or *argv* strings.

If a trace created with **Tcl_CreateTrace** is in effect, inline compilation of Tcl commands such as **if** and **while** is always disabled. There is no notification when a trace created with **Tcl_CreateTrace** is deleted. There is no way to be notified when the trace created by **Tcl_CreateTrace** is deleted. There is no way for the *proc* associated with a call to **Tcl_CreateTrace** to abort execution of *command*.

**KEYWORDS**

command, create, delete, interpreter, trace

Copyright ' 1989-1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 2002 by Kevin B. Kenny. All rights reserved.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
DString

NAME

Tcl_DStringInit, Tcl_DStringAppend, Tcl_DStringAppendElement, Tcl_DStringStartSublist,
Tcl_DStringEndSublist, Tcl_DStringLength, Tcl_DStringValue, Tcl_DStringSetLength, Tcl_DStringTrunc,
Tcl_DStringFree, Tcl_DStringResult, Tcl_DStringGetResult – manipulate dynamic strings

SYNOPSIS

#include <tcl.h>
Tcl_DStringInit(dsPtr)
char *
Tcl_DStringAppend(dsPtr, string, length)
char *
Tcl_DStringAppendElement(dsPtr, string)
Tcl_DStringStartSublist(dsPtr)
Tcl_DStringEndSublist(dsPtr)
int
Tcl_DStringLength(dsPtr)
char *
Tcl_DStringValue(dsPtr)
Tcl_DStringSetLength(dsPtr, newLength)
Tcl_DStringTrunc(dsPtr, newLength)
Tcl_DStringFree(dsPtr)
Tcl_DStringResult(interp, dsPtr)
Tcl_DStringGetResult(interp, dsPtr)

ARGUMENTS

Tcl_DString *dsPtr (in/out)
Pointer to structure that is used to manage a dynamic string.
CONST char *string (in)
Pointer to characters to add to dynamic string.
int length (in)
Number of characters from string to add to dynamic string. If −1, add all characters up to null
terminating character.
int newLength (in)
New length for dynamic string, not including null terminating character.
Tcl_Interp *interp (in/out)
Interpreter whose result is to be set from or moved to the dynamic string.
DESCRIPTION

Dynamic strings provide a mechanism for building up arbitrarily long strings by gradually appending information. If the dynamic string is short then there will be no memory allocation overhead; as the string gets larger, additional space will be allocated as needed.

Tcl_DStringInit initializes a dynamic string to zero length. The Tcl_DString structure must have been allocated by the caller. No assumptions are made about the current state of the structure; anything already in it is discarded. If the structure has been used previously, Tcl_DStringFree should be called first to free up any memory allocated for the old string.

Tcl_DStringAppend adds new information to a dynamic string, allocating more memory for the string if needed. If length is less than zero then everything in string is appended to the dynamic string; otherwise length specifies the number of bytes to append. Tcl_DStringAppend returns a pointer to the characters of the new string. The string can also be retrieved from the string field of the Tcl_DString structure.

Tcl_DStringAppendElement is similar to Tcl_DStringAppend except that it doesn't take a length argument (it appends all of string) and it converts the string to a proper list element before appending. Tcl_DStringAppendElement adds a separator space before the new list element unless the new list element is the first in a list or sub–list (i.e. either the current string is empty, or it contains the single character "", or the last two characters of the current string are " "). Tcl_DStringAppendElement returns a pointer to the characters of the new string.

Tcl_DStringStartSublist and Tcl_DStringEndSublist can be used to create nested lists. To append a list element that is itself a sublist, first call Tcl_DStringStartSublist, then call Tcl_DStringAppendElement for each of the elements in the sublist, then call Tcl_DStringEndSublist to end the sublist. Tcl_DStringStartSublist appends a space character if needed, followed by an open brace; Tcl_DStringEndSublist appends a close brace. Lists can be nested to any depth.

Tcl_DStringLength is a macro that returns the current length of a dynamic string (not including the terminating null character). Tcl_DStringValue is a macro that returns a pointer to the current contents of a dynamic string.

Tcl_DStringSetLength changes the length of a dynamic string. If newLength is less than the string's current length, then the string is truncated. If newLength is greater than the string's current length, then the string will become longer and new space will be allocated for the string if needed. However, Tcl_DStringSetLength will not initialize the new space except to provide a terminating null character; it is up to the caller to fill in the new space. Tcl_DStringSetLength does not free up the string's storage space even if the string is truncated to zero length, so Tcl_DStringFree will still need to be called.

Tcl_DStringTrunc changes the length of a dynamic string. This procedure is now deprecated. Tcl_DStringSetLength should be used instead.

Tcl_DStringFree should be called when you're finished using the string. It frees up any memory that was allocated for the string and reinitializes the string's value to an empty string.
Tcl_DStringResult sets the result of interp to the value of the dynamic string given by dsPtr. It does this by moving a pointer from dsPtr to the interpreter's result. This saves the cost of allocating new memory and copying the string. Tcl_DStringResult also reinitializes the dynamic string to an empty string.

Tcl_DStringGetResult does the opposite of Tcl_DStringResult. It sets the value of dsPtr to the result of interp and it clears interp's result. If possible it does this by moving a pointer rather than by copying the string.

**KEYWORDS**
append, dynamic string, free, result

---

Copyright ' 1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
DetachPids

NAME

Tcl_DetachPids, Tcl_ReapDetachedProcs, Tcl_WaitPid – manage child processes in background

SYNOPSIS

#include <tcl.h>
Tcl_DetachPids(numPids, pidPtr)
Tcl_ReapDetachedProcs()
Tcl_Pid
Tcl_WaitPid(pid, statPtr, options)

ARGUMENTS

int numPids (in)
    Number of process ids contained in the array pointed to by pidPtr.
int *pidPtr (in)
    Address of array containing numPids process ids.
Tcl_Pid pid (in)
    The id of the process (pipe) to wait for.
int* statPtr (out)
    The result of waiting on a process (pipe). Either 0 or ECHILD.
int options ()
    The options controlling the wait. WNOHANG specifies not to wait when checking the process.

DESCRIPTION

Tcl_DetachPids and Tcl_ReapDetachedProcs provide a mechanism for managing subprocesses that are running in background. These procedures are needed because the parent of a process must eventually invoke the waitpid kernel call (or one of a few other similar kernel calls) to wait for the child to exit. Until the parent waits for the child, the child’s state cannot be completely reclaimed by the system. If a parent continually creates children and doesn’t wait on them, the system’s process table will eventually overflow, even if all the children have exited.

Tcl_DetachPids may be called to ask Tcl to take responsibility for one or more processes whose process ids are contained in the pidPtr array passed as argument. The caller presumably has started these processes running in background and doesn't want to have to deal with them again.

Tcl_ReapDetachedProcs invokes the waitpid kernel call on each of the background processes so that its state can be cleaned up if it has exited. If the process hasn't exited yet, Tcl_ReapDetachedProcs doesn't wait for it to exit; it will check again the next time it is invoked. Tcl automatically calls Tcl_ReapDetachedProcs each time the exec command is executed, so in most cases it isn't necessary for any code outside of Tcl to
invoke `Tcl_ReapDetachedProcs`. However, if you call `Tcl_DetachPids` in situations where the `exec` command may never get executed, you may wish to call `Tcl_ReapDetachedProcs` from time to time so that background processes can be cleaned up.

`Tcl_WaitPid` is a thin wrapper around the facilities provided by the operating system to wait on the end of a spawned process and to check whether spawned process is still running. It is used by `Tcl_ReapDetachedProcs` and the channel system to portably access the operating system.

**KEYWORDS**

background, child, detach, process, wait

---

**Copyright**  1989-1993 The Regents of the University of California.
**Copyright**  1994-1996 Sun Microsystems, Inc.
**Copyright**  1995-1997 Roger E. Critchlow Jr.
DoOneEvent

NAME

Tcl_DoOneEvent – wait for events and invoke event handlers

SYNOPSIS

#include <tcl.h>
int Tcl_DoOneEvent(int flags)

ARGUMENTS

int flags (in)

This parameter is normally zero. It may be an OR-ed combination of any of the following flag bits:
TCL_WINDOW_EVENTS, TCL_FILE_EVENTS, TCL_TIMER_EVENTS, TCL_IDLE_EVENTS, TCL_ALL_EVENTS, or TCL_DONT_WAIT.

DESCRIPTION

This procedure is the entry point to Tcl's event loop; it is responsible for waiting for events and dispatching
event handlers created with procedures such as Tk_CreateEventHandler, Tcl_CreateFileHandler,
Tcl_CreateTimerHandler, and Tcl_DoWhenIdle. Tcl_DoOneEvent checks to see if events are already
present on the Tcl event queue; if so, it calls the handler(s) for the first (oldest) event, removes it from the
queue, and returns. If there are no events ready to be handled, then Tcl_DoOneEvent checks for new events
from all possible sources. If any are found, it puts all of them on Tcl's event queue, calls handlers for the first
event on the queue, and returns. If no events are found, Tcl_DoOneEvent checks for Tcl_DoWhenIdle
callbacks; if any are found, it invokes all of them and returns. Finally, if no events or idle callbacks have been
found, then Tcl_DoOneEvent sleeps until an event occurs; then it adds any new events to the Tcl event queue, calls handlers for the first event, and returns. The normal return value is 1 to signify that some event was processed (see below for other alternatives).

If the flags argument to Tcl_DoOneEvent is non-zero, it restricts the kinds of events that will be processed by Tcl_DoOneEvent. Flags may be an OR-ed combination of any of the following bits:

- **TCL_WINDOW_EVENTS** – Process window system events.
- **TCL_FILE_EVENTS** – Process file events.
- **TCL_TIMER_EVENTS** – Process timer events.
- **TCL_IDLE_EVENTS** – Process idle callbacks.
- **TCL_ALL_EVENTS** – Process all kinds of events: equivalent to OR-ing together all of the above flags or specifying none of them.
- **TCL_DONT_WAIT** – Don't sleep: process only events that are ready at the time of the call.

If any of the flags TCL_WINDOW_EVENTS, TCL_FILE_EVENTS, TCL_TIMER_EVENTS, or TCL_IDLE_EVENTS is set, then the only events that will be considered are those for which flags are set. Setting none of these flags is equivalent to the value TCL_ALL_EVENTS, which causes all event types to be processed. If an application has defined additional event sources with Tcl_CreateEventSource, then additional flag values may also be valid, depending on those event sources.

The TCL_DONT_WAIT flag causes Tcl_DoOneEvent not to put the process to sleep: it will check for events but if none are found then it returns immediately with a return value of 0 to indicate that no work was done. Tcl_DoOneEvent will also return 0 without doing anything if the only alternative is to block forever (this can happen, for example, if flags is TCL_IDLE_EVENTS and there are no Tcl_DoWhenIdle callbacks pending, or if no event handlers or timer handlers exist).

Tcl_DoOneEvent may be invoked recursively. For example, it is possible to invoke Tcl_DoOneEvent recursively from a handler called by Tcl_DoOneEvent. This sort of operation is useful in some modal situations, such as when a notification dialog has been popped up and an application wishes to wait for the user to click a button in the dialog before doing anything else.

**KEYWORDS**

callback, event, handler, idle, timer

---

Copyright ' 1990-1992 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
DoWhenIdle

NAME

Tcl_DoWhenIdle, Tcl_CancelIdleCall – invoke a procedure when there are no pending events

SYNOPSIS

#include <tcl.h>
Tcl_DoWhenIdle(proc, clientData)
Tcl_CancelIdleCall(proc, clientData)

ARGUMENTS

Tcl_IdleProc *proc (in)
    Procedure to invoke.
ClientData clientData (in)
    Arbitrary one-word value to pass to proc.

DESCRIPTION

Tcl_DoWhenIdle arranges for proc to be invoked when the application becomes idle. The application is considered to be idle when Tcl_DoOneEvent has been called, couldn't find any events to handle, and is about to go to sleep waiting for an event to occur. At this point all pending Tcl_DoWhenIdle handlers are invoked. For each call to Tcl_DoWhenIdle there will be a single call to proc; after proc is invoked the handler is automatically removed. Tcl_DoWhenIdle is only usable in programs that use Tcl_DoOneEvent to dispatch events.

Proc should have arguments and result that match the type Tcl_IdleProc:

typedef void Tcl_IdleProc(ClientData clientData);

The clientData parameter to proc is a copy of the clientData argument given to Tcl_DoWhenIdle. Typically, clientData points to a data structure containing application-specific information about what proc should do.

Tcl_CancelIdleCall may be used to cancel one or more previous calls to Tcl_DoWhenIdle: if there is a Tcl_DoWhenIdle handler registered for proc and clientData, then it is removed without invoking it. If there is more than one handler on the idle list that refers to proc and clientData, all of the handlers are removed. If no existing handlers match proc and clientData then nothing happens.

Tcl_DoWhenIdle is most useful in situations where (a) a piece of work will have to be done but (b) it's possible that something will happen in the near future that will change what has to be done or require something different to be done. Tcl_DoWhenIdle allows the actual work to be deferred until all pending events have been processed. At this point the exact work to be done will presumably be known and it can be
done exactly once.

For example, **Tcl_DoWhenIdle** might be used by an editor to defer display updates until all pending commands have been processed. Without this feature, redundant redisplay might occur in some situations, such as the processing of a command file.

**BUGS**

At present it is not safe for an idle callback to reschedule itself continuously. This will interact badly with certain features of Tk that attempt to wait for all idle callbacks to complete. If you would like for an idle callback to reschedule itself continuously, it is better to use a timer handler with a zero timeout period.

**KEYWORDS**

callback, defer, idle callback
DoubleObj

NAME

Tcl_NewDoubleObj, Tcl_SetDoubleObj, Tcl_GetDoubleFromObj – manipulate Tcl objects as floating-point values

SYNOPSIS

#include <tcl.h>
Tcl_Obj *
Tcl_NewDoubleObj(doubleValue)
Tcl_SetDoubleObj(objPtr, doubleValue)
int
Tcl_GetDoubleFromObj(interp, objPtr, doublePtr)

ARGUMENTS

double doubleValue (in)
A double-precision floating point value used to initialize or set a double object.

Tcl_Obj *objPtr (in/out)
For Tcl_SetDoubleObj, this points to the object to be converted to double type. For
Tcl_GetDoubleFromObj, this refers to the object from which to get a double value; if objPtr does
not already point to a double object, an attempt will be made to convert it to one.

Tcl_Interp *interp (in/out)
If an error occurs during conversion, an error message is left in the interpreter's result object unless
interp is NULL.

double *doublePtr (out)
Points to place to store the double value obtained from objPtr.

DESCRIPTION

These procedures are used to create, modify, and read double Tcl objects from C code. Tcl_NewDoubleObj
and Tcl_SetDoubleObj will create a new object of double type or modify an existing object to have double
type. Both of these procedures set the object to have the double-precision floating point value given by
doubleValue; Tcl_NewDoubleObj returns a pointer to a newly created object with reference count zero. Both
procedures set the object's type to be double and assign the double value to the object's internal representation
doubleValue member. Tcl_SetDoubleObj invalidates any old string representation and, if the object is not
already a double object, frees any old internal representation.

Tcl_GetDoubleFromObj attempts to return a double value from the Tcl object objPtr. If the object is not
already a double object, it will attempt to convert it to one. If an error occurs during conversion, it returns
TCL_ERROR and leaves an error message in the interpreter's result object unless interp is NULL.
Otherwise, it returns TCL_OK and stores the double value in the address given by doublePtr. If the object is
not already a double object, the conversion will free any old internal representation.

SEE ALSO

_Tcl_NewObj_, _Tcl_DecrRefCount_, _Tcl_IncrRefCount_, _Tcl_GetObjResult_

KEYWORDS

double, double object, double type, internal representation, object, object type, string representation

Copyright ’ 1996-1997 Sun Microsystems, Inc.
DumpActiveMemory

NAME

Tcl_DumpActiveMemory, Tcl_InitMemory, Tcl_ValidateAllMemory – Validated memory allocation interface.

SYNOPSIS

#include <tcl.h>
int
Tcl_DumpActiveMemory(fileName)
void
Tcl_InitMemory(interp)
void
Tcl_ValidateAllMemory(fileName, line)

ARGUMENTS

Tcl_Interp * interp (in)
Tcl interpreter in which to add commands.
CONST char * fileName (in)
For Tcl_DumpActiveMemory, name of the file to which memory information will be written. For
Tcl_ValidateAllMemory, name of the file from which the call is being made (normally __FILE__).
int line (in)
Line number at which the call to Tcl_ValidateAllMemory is made (normally __LINE__).

DESCRIPTION

These functions provide access to Tcl memory debugging information. They are only functional when Tcl has
been compiled with TCL_MEM_DEBUG defined at compile–time. When TCL_MEM_DEBUG is not
defined, these functions are all no–ops.

Tcl_DumpActiveMemory will output a list of all currently allocated memory to the specified file. The
information output for each allocated block of memory is: starting and ending addresses (excluding guard
zone), size, source file where calloc was called to allocate the block and line number in that file. It is
especially useful to call Tcl_DumpActiveMemory after the Tcl interpreter has been deleted.

Tcl_InitMemory adds the Tcl_memory command to the interpreter given by interp. Tcl_InitMemory is
called by Tcl_Main.

Tcl_ValidateAllMemory forces a validation of the guard zones of all currently allocated blocks of memory.
Normally validation of a block occurs when its freed, unless full validation is enabled, in which case
validation of all blocks occurs when _ckalloc and _ckfree are called. This function forces the validation to occur at any point.

SEE ALSO

TCL_MEM_DEBUG, memory

KEYWORDS

memory, debug

Copyright ' 1992−1999 Karl Lehenbauer and Mark Diekhans.
Copyright ' 2000 by Scriptics Corporation.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
Eval

NAME

Tcl_EvalObjEx, Tcl_EvalFile, Tcl_EvalObjv, Tcl_Eval, Tcl_EvalEx, Tcl_GlobalEval, Tcl_GlobalEvalObj, Tcl_VarEval, Tcl_VarEvalVA – execute Tcl scripts

SYNOPSIS

ARGUMENTS

DESCRIPTION

FLAG BITS

TCL_EVAL_DIRECT
TCL_EVAL_GLOBAL

MISCELLANEOUS DETAILS

KEYWORDS

NAME

Tcl_EvalObjEx, Tcl_EvalFile, Tcl_EvalObjv, Tcl_Eval, Tcl_EvalEx, Tcl_GlobalEval, Tcl_GlobalEvalObj, Tcl_VarEval, Tcl_VarEvalVA – execute Tcl scripts

SYNOPSIS

#include <tcl.h>
int
Tcl_EvalObjEx(interp, objPtr, flags)
int
Tcl_EvalFile(interp, fileName)
int
Tcl_EvalObjv(interp, objc, objv, flags)
int
Tcl_Eval(interp, script)
int
Tcl_EvalEx(interp, script, numBytes, flags)
int
Tcl_GlobalEval(interp, script)
int
Tcl_GlobalEvalObj(interp, objPtr)
int
Tcl_VarEval(interp, string, string, ... (char *) NULL)
int
Tcl_VarEvalVA(interp, argList)
ARGUMENTS

Tcl_Interp *interp (in)
   Interpreter in which to execute the script. The interpreter's result is modified to hold the result or error message from the script.

Tcl_Obj *objPtr (in)
   A Tcl object containing the script to execute.

int flags (in)
   ORed combination of flag bits that specify additional options. TCL_EVAL_GLOBAL and TCL_EVAL_DIRECT are currently supported.

CONST char *fileName (in)
   Name of a file containing a Tcl script.

int objc (in)
   The number of objects in the array pointed to by objPtr; this is also the number of words in the command.

Tcl_Obj **objv (in)
   Points to an array of pointers to objects; each object holds the value of a single word in the command to execute.

int numBytes (in)
   The number of bytes in script, not including any null terminating character. If −1, then all characters up to the first null byte are used.

CONST char *script (in)
   Points to first byte of script to execute (null−terminated and UTF−8).

char *string (in)
   String forming part of a Tcl script.

va_list argList (in)
   An argument list which must have been initialised using TCL_VARARGS_START, and cleared using va_end.

DESCRIPTION

The procedures described here are invoked to execute Tcl scripts in various forms. Tcl_EvalObjEx is the core procedure and is used by many of the others. It executes the commands in the script stored in objPtr until either an error occurs or the end of the script is reached. If this is the first time objPtr has been executed, its commands are compiled into bytecode instructions which are then executed. The bytecodes are saved in objPtr so that the compilation step can be skipped if the object is evaluated again in the future.

The return value from Tcl_EvalObjEx (and all the other procedures described here) is a Tcl completion code with one of the values TCL_OK, TCL_ERROR, TCL_RETURN, TCL_BREAK, or TCL_CONTINUE, or possibly some other integer value originating in an extension. In addition, a result value or error message is left in interp's result; it can be retrieved using Tcl_GetObjResult.

Tcl_EvalFile reads the file given by fileName and evaluates its contents as a Tcl script. It returns the same information as Tcl_EvalObjEx. If the file couldn't be read then a Tcl error is returned to describe why the file couldn't be read. The eofchar for files is \32' (^Z) for all platforms. If you require a ``^Z'' in code for string
comparison, you can use `\032` or `\u001a`, which will be safely substituted by the Tcl interpreter into `\^Z`.

Tcl_EvalObjv executes a single pre–parsed command instead of a script. The obj and objv arguments contain the values of the words for the Tcl command, one word in each object in objv. Tcl_EvalObjv evaluates the command and returns a completion code and result just like Tcl_EvalObjEx.

Tcl_Eval is similar to Tcl_EvalObjEx except that the script to be executed is supplied as a string instead of an object and no compilation occurs. The string should be a proper UTF–8 string as converted by Tcl_ExternalToUtfDString or Tcl_ExternalToUtf when it is known to possibly contain upper ASCII characters who's possible combinations might be a UTF–8 special code. The string is parsed and executed directly (using Tcl_EvalObjv) instead of compiling it and executing the bytecodes. In situations where it is known that the script will never be executed again, Tcl_Eval may be faster than Tcl_EvalObjEx. Tcl_Eval returns a completion code and result just like Tcl_EvalObjEx. Note: for backward compatibility with versions before Tcl 8.0, Tcl_Eval copies the object result in interp to interp–>result (use is deprecated) where it can be accessed directly. This makes Tcl_Eval somewhat slower than Tcl_EvalEx, which doesn't do the copy.

Tcl_EvalEx is an extended version of Tcl_Eval that takes additional arguments numBytes and flags. For the efficiency reason given above, Tcl_EvalEx is generally preferred over Tcl_Eval.

Tcl_GlobalEval and Tcl_GlobalEvalObj are older procedures that are now deprecated. They are similar to Tcl_EvalEx and Tcl_EvalObjEx except that the script is evaluated in the global namespace and its variable context consists of global variables only (it ignores any Tcl procedures that are active). These functions are equivalent to using the TCL_EVAL_GLOBAL flag (see below).

Tcl_VarEval takes any number of string arguments of any length, concatenates them into a single string, then calls Tcl_Eval to execute that string as a Tcl command. It returns the result of the command and also modifies interp–>result in the same way as Tcl_Eval. The last argument to Tcl_VarEval must be NULL to indicate the end of arguments. Tcl_VarEval is now deprecated.

Tcl_VarEvalVA is the same as Tcl_VarEval except that instead of taking a variable number of arguments it takes an argument list. Like Tcl_VarEval, Tcl_VarEvalVA is deprecated.

**FLAG BITS**

Any ORed combination of the following values may be used for the flags argument to procedures such as Tcl_EvalObjEx:

**TCL_EVAL_DIRECT**

This flag is only used by Tcl_EvalObjEx; it is ignored by other procedures. If this flag bit is set, the script is not compiled to bytecodes; instead it is executed directly as is done by Tcl_EvalEx. The TCL_EVAL_DIRECT flag is useful in situations where the contents of an object are going to change immediately, so the bytecodes won't be reused in a future execution. In this case, it's faster to execute the script directly.
TCL_EVAL_GLOBAL

If this flag is set, the script is processed at global level. This means that it is evaluated in the global namespace and its variable context consists of global variables only (it ignores any Tcl procedures at are active).

MISCELLANEOUS DETAILS

During the processing of a Tcl command it is legal to make nested calls to evaluate other commands (this is how procedures and some control structures are implemented). If a code other than TCL_OK is returned from a nested Tcl_EvalObjEx invocation, then the caller should normally return immediately, passing that same return code back to its caller, and so on until the top−level application is reached. A few commands, like for, will check for certain return codes, like TCL_BREAK and TCL_CONTINUE, and process them specially without returning.

Tcl_EvalObjEx keeps track of how many nested Tcl_EvalObjEx invocations are in progress for interp. If a code of TCL_RETURN, TCL_BREAK, or TCL_CONTINUE is about to be returned from the topmost Tcl_EvalObjEx invocation for interp, it converts the return code to TCL_ERROR and sets interp's result to an error message indicating that the return, break, or continue command was invoked in an inappropriate place. This means that top−level applications should never see a return code from Tcl_EvalObjEx other then TCL_OK or TCL_ERROR.

KEYWORDS

execute, file, global, object, result, script

Copyright ' 1989−1993 The Regents of the University of California.
Copyright ' 1994−1997 Sun Microsystems, Inc.
Copyright ' 2000 Scriptics Corporation.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
Exit

NAME

Tcl_Exit, Tcl_Finalize, Tcl_CreateExitHandler, Tcl_DeleteExitHandler, Tcl_ExitThread, Tcl_FinalizeThread, Tcl_CreateThreadExitHandler, Tcl_DeleteThreadExitHandler – end the application or thread (and invoke exit handlers)

SYNOPSIS

#include <tcl.h>
Tcl_Exit(status)
Tcl_Finalize()
Tcl_CreateExitHandler(proc, clientData)
Tcl_DeleteExitHandler(proc, clientData)
Tcl_ExitThread(status)
Tcl_FinalizeThread()
Tcl_CreateThreadExitHandler(proc, clientData)
Tcl_DeleteThreadExitHandler(proc, clientData)

ARGUMENTS

int status (in)
    Provides information about why the application or thread exited. Exact meaning may be
    platform-specific. 0 usually means a normal exit, any nonzero value usually means that an error
    occurred.

Tcl_ExitProc *proc (in)
    Procedure to invoke before exiting application.

ClientData clientData (in)
    Arbitrary one-word value to pass to proc.

DESCRIPTION

The procedures described here provide a graceful mechanism to end the execution of a Tcl application. Exit handlers are invoked to cleanup the application's state before ending the execution of Tcl code.

Invoke Tcl_Exit to end a Tcl application and to exit from this process. This procedure is invoked by the exit command, and can be invoked anywhere else to terminate the application. No-one should ever invoke the exit system procedure directly; always invoke Tcl_Exit instead, so that it can invoke exit handlers. Note that if other code invokes exit system procedure directly, or otherwise causes the application to terminate without calling Tcl_Exit, the exit handlers will not be run. Tcl_Exit internally invokes the exit system call, thus it never returns control to its caller.
**Tcl_Finalize** is similar to **Tcl_Exit** except that it does not exit from the current process. It is useful for cleaning up when a process is finished using **Tcl** but wishes to continue executing, and when **Tcl** is used in a dynamically loaded extension that is about to be unloaded. On some systems **Tcl** is automatically notified when it is being unloaded, and it calls **Tcl_Finalize** internally; on these systems it not necessary for the caller to explicitly call **Tcl_Finalize**. However, to ensure portability, your code should always invoke **Tcl_Finalize** when **Tcl** is being unloaded, to ensure that the code will work on all platforms. **Tcl_Finalize** can be safely called more than once.

**Tcl_Finalize** is used to terminate the current thread and invoke per-thread exit handlers. This finalization is done by **Tcl_FinalizeThread**, which you can call if you just want to clean up per-thread state and invoke the thread exit handlers. **Tcl_Finalize** calls **Tcl_FinalizeThread** for the current thread automatically.

**Tcl_Finalize** and **Tcl_FinalizeThread** execute all registered exit handlers, in reverse order from the order in which they were registered. This matches the natural order in which extensions are loaded and unloaded; if extension A loads extension B, it usually unloads B before it itself is unloaded. If extension A registers its exit handlers before loading extension B, this ensures that any exit handlers for B will be executed before the exit handlers for A.

**Tcl_Finalize** and **Tcl_FinalizeThread** call **Tcl_FinalizeThread** and the thread exit handlers after the process-wide exit handlers. This is because thread finalization shuts down the I/O channel system, so any attempt at I/O by the global exit handlers will vanish into the bitbucket.

**KEYWORDS**

callback, cleanup, dynamic loading, end application, exit, unloading, thread

---

Copyright © 1995-1996 Sun Microsystems, Inc.
Copyright © 1995-1997 Roger E. Critchlow Jr.
ExprLong

NAME

Tcl_ExprLong, Tcl_ExprDouble, Tcl_ExprBoolean, Tcl_ExprString − evaluate an expression

SYNOPSIS

#include <tcl.h>
int
Tcl_ExprLong(interp, string, longPtr)
int
Tcl_ExprDouble(interp, string, doublePtr)
int
Tcl_ExprBoolean(interp, string, booleanPtr)
int
Tcl_ExprString(interp, string)

ARGUMENTS

Tcl_Interp *interp (in)
    Interpreter in whose context to evaluate string or objPtr.
CONST char *string (in)
    Expression to be evaluated.
long *longPtr (out)
    Pointer to location in which to store the integer value of the expression.
int *doublePtr (out)
    Pointer to location in which to store the floating−point value of the expression.
int *booleanPtr (out)
    Pointer to location in which to store the 0/1 boolean value of the expression.

DESCRIPTION

These four procedures all evaluate the expression given by the string argument and return the result in one of four different forms. The expression can have any of the forms accepted by the expr command. Note that these procedures have been largely replaced by the object−based procedures Tcl_ExprLongObj, Tcl_ExprDoubleObj, Tcl_ExprBooleanObj, and Tcl_ExprObj. Those object−based procedures evaluate an expression held in a Tcl object instead of a string. The object argument can retain an internal representation that is more efficient to execute.

The interp argument refers to an interpreter used to evaluate the expression (e.g. for variables and nested Tcl commands) and to return error information.
For all of these procedures the return value is a standard Tcl result: **TCL_OK** means the expression was successfully evaluated, and **TCL_ERROR** means that an error occurred while evaluating the expression. If **TCL_ERROR** is returned then the interpreter's result will hold a message describing the error. If an error occurs while executing a Tcl command embedded in the expression then that error will be returned.

If the expression is successfully evaluated, then its value is returned in one of four forms, depending on which procedure is invoked. **Tcl_ExprLong** stores an integer value at *longPtr. If the expression's actual value is a floating–point number, then it is truncated to an integer. If the expression's actual value is a non–numeric string then an error is returned.

**Tcl_ExprDouble** stores a floating–point value at *doublePtr. If the expression's actual value is an integer, it is converted to floating–point. If the expression's actual value is a non–numeric string then an error is returned.

**Tcl_ExprBoolean** stores a 0/1 integer value at *booleanPtr. If the expression's actual value is an integer or floating–point number, then they store 0 at *booleanPtr if the value was zero and 1 otherwise. If the expression's actual value is a non–numeric string then it must be one of the values accepted by **Tcl_GetBoolean** such as ```yes``` or ```no```, or else an error occurs.

**Tcl_ExprString** returns the value of the expression as a string stored in the interpreter's result. If the expression's actual value is an integer then **Tcl_ExprString** converts it to a string using **sprintf** with a ```%d``` converter. If the expression's actual value is a floating–point number, then **Tcl_ExprString** calls **Tcl_PrintDouble** to convert it to a string.

**SEE ALSO**

**Tcl_ExprLongObj, Tcl_ExprDoubleObj, Tcl_ExprBooleanObj, Tcl_ExprObj**

**KEYWORDS**

boolean, double, evaluate, expression, integer, object, string

---

**Copyright** ' 1989-1993 The Regents of the University of California.  
**Copyright** ' 1994-1997 Sun Microsystems, Inc.  
**Copyright** ' 1995-1997 Roger E. Critchlow Jr.
ExprLongObj

NAME

Tcl_ExprLongObj, Tcl_ExprDoubleObj, Tcl_ExprBooleanObj, Tcl_ExprObj − evaluate an expression

SYNOPSIS

#include <tcl.h>

int Tcl_ExprLongObj(interp, objPtr, longPtr)

int Tcl_ExprDoubleObj(interp, objPtr, doublePtr)

int Tcl_ExprBooleanObj(interp, objPtr, booleanPtr)

int Tcl_ExprObj(interp, objPtr, resultPtrPtr)

ARGUMENTS

Tcl_Interp *interp (in)
Interpreter in whose context to evaluate string or objPtr.

Tcl_Obj *objPtr (in)
Pointer to an object containing the expression to evaluate.

long *longPtr (out)
Pointer to location in which to store the integer value of the expression.

int *doublePtr (out)
Pointer to location in which to store the floating−point value of the expression.

int *booleanPtr (out)
Pointer to location in which to store the 0/1 boolean value of the expression.

Tcl_Obj **resultPtrPtr (out)
Pointer to location in which to store a pointer to the object that is the result of the expression.

DESCRIPTION

These four procedures all evaluate an expression, returning the result in one of four different forms. The expression is given by the objPtr argument, and it can have any of the forms accepted by the expr command.

The interp argument refers to an interpreter used to evaluate the expression (e.g. for variables and nested Tcl commands) and to return error information.

For all of these procedures the return value is a standard Tcl result: TCL_OK means the expression was successfully evaluated, and TCL_ERROR means that an error occurred while evaluating the expression. If TCL_ERROR is returned, then a message describing the error can be retrieved using Tcl_GetObjResult.
an error occurs while executing a Tcl command embedded in the expression then that error will be returned.

If the expression is successfully evaluated, then its value is returned in one of four forms, depending on which procedure is invoked. **TclExprLongObj** stores an integer value at *longPtr. If the expression's actual value is a floating−point number, then it is truncated to an integer. If the expression's actual value is a non−numeric string then an error is returned.

**TclExprDoubleObj** stores a floating−point value at *doublePtr. If the expression's actual value is an integer, it is converted to floating−point. If the expression's actual value is a non−numeric string then an error is returned.

**TclExprBooleanObj** stores a 0/1 integer value at *booleanPtr. If the expression's actual value is an integer or floating−point number, then they store 0 at *booleanPtr if the value was zero and 1 otherwise. If the expression's actual value is a non−numeric string then it must be one of the values accepted by **TclGetBoolean** such as ``yes'' or ``no'', or else an error occurs.

If **TclExprObj** successfully evaluates the expression, it stores a pointer to the Tcl object containing the expression's value at *resultPtrPtr. In this case, the caller is responsible for calling **TclDecrRefCount** to decrement the object's reference count when it is finished with the object.

**SEE ALSO**

**TclExprLong**, **TclExprDouble**, **TclExprBoolean**, **TclExprString**, **TclGetObjResult**

**KEYWORDS**

boolean, double, evaluate, expression, integer, object, string

---

Copyright ' 1996-1997 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
FindExec

NAME

Tcl_FindExecutable, Tcl_GetNameOfExecutable – identify or return the name of the binary file containing the application

SYNOPSIS

#include <tcl.h>
void
Tcl_FindExecutable(argv0)
CONST char *
Tcl_GetNameOfExecutable()

ARGUMENTS

char *argv0 (in)
The first command-line argument to the program, which gives the application's name.

DESCRIPTION

The Tcl_FindExecutable procedure computes the full path name of the executable file from which the application was invoked and saves it for Tcl's internal use. The executable's path name is needed for several purposes in Tcl. For example, it is needed on some platforms in the implementation of the load command. It is also returned by the info nameofexecutable command.

On UNIX platforms this procedure is typically invoked as the very first thing in the application's main program; it must be passed argv[0] as its argument. It is important not to change the working directory before the invocation. Tcl_FindExecutable uses argv0 along with the PATH environment variable to find the application's executable, if possible. If it fails to find the binary, then future calls to info nameofexecutable will return an empty string.

Tcl_GetNameOfExecutable simply returns a pointer to the internal full path name of the executable file as computed by Tcl_FindExecutable. This procedure call is the C API equivalent to the info nameofexecutable command. NULL is returned if the internal full path name has not been computed or unknown.

KEYWORDS

binary, executable file

Copyright ' 1995-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
GetCwd

NAME

Tcl_GetCwd, Tcl_Chdir – manipulate the current working directory

SYNOPSIS

#include <tcl.h>
char *
Tcl_GetCwd(interp, bufferPtr)
int
Tcl_Chdir(path)

ARGUMENTS

Tcl_Interp *interp (in)
   Interpreter in which to report an error, if any.
Tcl_DString *bufferPtr (in/out)
   This dynamic string is used to store the current working directory. At the time of the call it should be
   uninitialized or free. The caller must eventually call Tcl_DStringFree to free up anything stored here.
char *path (in)
   File path in UTF−8 format.

DESCRIPTION

These procedures may be used to manipulate the current working directory for the application. They provide
C−level access to the same functionality as the Tcl pwd command.

Tcl_GetCwd returns a pointer to a string specifying the current directory, or NULL if the current directory
could not be determined. If NULL is returned, an error message is left in the interp's result. Storage for the
result string is allocated in bufferPtr; the caller must call Tcl_DStringFree() when the result is no longer
needed. The format of the path is UTF−8.

Tcl_Chdir changes the applications current working directory to the value specified in path. The format of
the passed in string must be UTF−8. The function returns −1 on error or 0 on success.

KEYWORDS

pwd

Copyright ' 1998−1999 Scriptics Corporation
Copyright ' 1995−1997 Roger E. Critchlow Jr.
Encoding

NAME

Tcl_GetEncoding, Tcl_FreeEncoding, Tcl_ExternalToUtfDString, Tcl_ExternalToUtf,
Tcl_UtfToExternalDString, Tcl_UtfToExternal, Tcl_WinTCharToUtf, Tcl_WinUtfToTChar,
Tcl_GetEncodingName, Tcl_SetSystemEncoding, Tcl_GetEncodingNames,
Tcl_CreateEncoding, Tcl_GetDefaultEncodingDir, Tcl_SetDefaultEncodingDir – procedures
for creating and using encodings.

SYNOPSIS

ARGUMENTS

INTRODUCTION

DESCRIPTION

TCL_OK

TCL_CONVERT_NOSPACE

TCL_CONVERT_MULTIBYTE

TCL_CONVERT_SYNTAX

TCL_CONVERT_UNKNOWN

ENCODING FILES

[1] S

[2] D

[3] M

[4] E

KEYWORDS

NAME

Tcl_GetEncoding, Tcl_FreeEncoding, Tcl_ExternalToUtfDString, Tcl_ExternalToUtf,
Tcl_UtfToExternalDString, Tcl_UtfToExternal, Tcl_WinTCharToUtf, Tcl_WinUtfToTChar,
Tcl_GetEncodingName, Tcl_SetSystemEncoding, Tcl_GetEncodingNames, Tcl_CreateEncoding,
Tcl_GetDefaultEncodingDir, Tcl_SetDefaultEncodingDir – procedures for creating and using encodings.

SYNOPSIS

#include <tcl.h>

Tcl_Encoding

Tcl_GetEncoding(interp, name)

void

Tcl_FreeEncoding(encoding)

char *

Tcl_ExternalToUtfDString(encoding, src, srcLen, dstPtr)

int

Tcl_ExternalToUtf(interp, encoding, src, srcLen, flags, statePtr, dst, dstLen, srcReadPtr, dstWrotePtr,
dstCharsPtr)
char *
Tcl_UtfToExternalDString(encoding, src, srcLen, dstPtr)
int
Tcl_UtfToExternal(interp, encoding, src, srcLen, flags, statePtr, dst, dstLen, srcReadPtr, dstWrotePtr, 
dstCharsPtr)
char *
Tcl_WinTCharToUtf(tsrc, srcLen, dstPtr)
TCHAR *
Tcl_WinUtfToTChar(src, srcLen, dstPtr)
CONST char *
Tcl_GetEncodingName(encoding)
int
Tcl_SetSystemEncoding(interp, name)
void
Tcl_GetEncodingNames(interp)
Tcl_Encoding
Tcl_CreateEncoding(typePtr)
CONST char *
Tcl_GetDefaultEncodingDir(void)
void
Tcl_SetDefaultEncodingDir(path)

ARGUMENTS

Tcl_Interp *interp (in)
Interpreter to use for error reporting, or NULL if no error reporting is desired.
CONST char *name (in)
Name of encoding to load.
Tcl_Encoding encoding (in)
The encoding to query, free, or use for converting text. If encoding is NULL, the current system
encoding is used.
CONST char *src (in)
For the Tcl_ExternalToUtf functions, an array of bytes in the specified encoding that are to be
converted to UTF−8. For the Tcl_UtfToExternal and Tcl_WinUtfToTChar functions, an array of
UTF−8 characters to be converted to the specified encoding.
CONST TCHAR *tsrc (in)
An array of Windows TCHAR characters to convert to UTF−8.
int srcLen (in)
Length of src or tsrc in bytes. If the length is negative, the encoding−specific length of the string is
used.
Tcl_DString *dstPtr (out)
Pointer to an uninitialized or free Tcl_DString in which the converted result will be stored.
int flags (in)

Various flag bits OR-ed together. TCL_ENCODING_START signifies that the source buffer is the first block in a (potentially multi-block) input stream, telling the conversion routine to reset to an initial state and perform any initialization that needs to occur before the first byte is converted. TCL_ENCODING_END signifies that the source buffer is the last block in a (potentially multi-block) input stream, telling the conversion routine to perform any finalization that needs to occur after the last byte is converted and then to reset to an initial state. TCL_ENCODING_STOPONERROR signifies that the conversion routine should return immediately upon reading a source character that doesn't exist in the target encoding; otherwise a default fallback character will automatically be substituted.

Tcl_EncodingState *statePtr (in/out)

Used when converting a (generally long or indefinite length) byte stream in a piece by piece fashion. The conversion routine stores its current state in *statePtr after src (the buffer containing the current piece) has been converted; that state information must be passed back when converting the next piece of the stream so the conversion routine knows what state it was in when it left off at the end of the last piece. May be NULL, in which case the value specified for flags is ignored and the source buffer is assumed to contain the complete string to convert.

char *dst (out)

Buffer in which the converted result will be stored. No more than dstLen bytes will be stored in dst.

int dstLen (in)

The maximum length of the output buffer dst in bytes.

int *srcReadPtr (out)

Filled with the number of bytes from src that were actually converted. This may be less than the original source length if there was a problem converting some source characters. May be NULL.

int *dstWrotePtr (out)

Filled with the number of bytes that were actually stored in the output buffer as a result of the conversion. May be NULL.

int *dstCharsPtr (out)

Filled with the number of characters that correspond to the number of bytes stored in the output buffer. May be NULL.

Tcl_EncodingType *typePtr (in)

Structure that defines a new type of encoding.

CONST char *path (in)

A path to the location of the encoding file.

INTRODUCTION

These routines convert between Tcl's internal character representation, UTF-8, and character representations used by various operating systems or file systems, such as Unicode, ASCII, or Shift-JIS. When operating on strings, such as such as obtaining the names of files or displaying characters using international fonts, the strings must be translated into one or possibly multiple formats that the various system calls can use. For instance, on a Japanese Unix workstation, a user might obtain a filename represented in the EUC-JP file encoding and then translate the characters to the jsix0208 font encoding in order to display the filename in a Tk widget. The purpose of the encoding package is to help bridge the translation gap. UTF-8 provides an intermediate staging ground for all the various encodings. In the example above, text would be translated into
UTF–8 from whatever file encoding the operating system is using. Then it would be translated from UTF–8 into whatever font encoding the display routines require.

Some basic encodings are compiled into Tcl. Others can be defined by the user or dynamically loaded from encoding files in a platform–independent manner.

DESCRIPTION

Tcl_GetEncoding finds an encoding given its name. The name may refer to a builtin Tcl encoding, a user–defined encoding registered by calling Tcl_CreateEncoding, or a dynamically–loadable encoding file. The return value is a token that represents the encoding and can be used in subsequent calls to procedures such as Tcl_GetEncodingName, Tcl_FreeEncoding, and Tcl_UtfToExternal. If the name did not refer to any known or loadable encoding, NULL is returned and an error message is returned in interp.

The encoding package maintains a database of all encodings currently in use. The first time name is seen, Tcl_GetEncoding returns an encoding with a reference count of 1. If the same name is requested further times, then the reference count for that encoding is incremented without the overhead of allocating a new encoding and all its associated data structures.

When an encoding is no longer needed, Tcl_FreeEncoding should be called to release it. When an encoding is no longer in use anywhere (i.e., it has been freed as many times as it has been gotten) Tcl_FreeEncoding will release all storage the encoding was using and delete it from the database.

Tcl_ExternalToUtfDString converts a source buffer src from the specified encoding into UTF–8. The converted bytes are stored in dstPtr, which is then null–terminated. The caller should eventually call Tcl_DStringFree to free any information stored in dstPtr. When converting, if any of the characters in the source buffer cannot be represented in the target encoding, a default fallback character will be used. The return value is a pointer to the value stored in the DString.

Tcl_ExternalToUtf converts a source buffer src from the specified encoding into UTF–8. Up to srcLen bytes are converted from the source buffer and up to dstLen converted bytes are stored in dst. In all cases, *srcReadPtr is filled with the number of bytes that were successfully converted from src and *dstWrotePtr is filled with the corresponding number of bytes that were stored in dst. The return value is one of the following:

- **TCL_OK**
  All bytes of src were converted.

- **TCL_CONVERT_NOSPACE**
  The destination buffer was not large enough for all of the converted data; as many characters as could fit were converted though.

- **TCL_CONVERT_MULTIBYTE**
  The last few bytes in the source buffer were the beginning of a multibyte sequence, but more bytes were needed to complete this sequence. A subsequent call to the conversion routine should pass a buffer containing the unconverted bytes that remained in src plus some further bytes from the source stream to properly convert the formerly split–up multibyte sequence.

- **TCL_CONVERT_SYNTAX**
The source buffer contained an invalid character sequence. This may occur if the input stream has been damaged or if the input encoding method was misidentified.

**TCL_CONVERT_UNKNOWN**

The source buffer contained a character that could not be represented in the target encoding and TCL_ENCODING_STOPONERROR was specified.

**Tcl_UtfToExternalDString** converts a source buffer src from UTF−8 into the specified encoding. The converted bytes are stored in dstPtr, which is then terminated with the appropriate encoding−specific null. The caller should eventually call **Tcl_DStringFree** to free any information stored in dstPtr. When converting, if any of the characters in the source buffer cannot be represented in the target encoding, a default fallback character will be used. The return value is a pointer to the value stored in the DString.

**Tcl_UtfToExternal** converts a source buffer src from UTF−8 into the specified encoding. Up to srcLen bytes are converted from the source buffer and up to dstLen converted bytes are stored in dst. In all cases, *srcReadPtr is filled with the number of bytes that were successfully converted from src and *dstWrotePtr is filled with the corresponding number of bytes that were stored in dst. The return values are the same as the return values for Tcl_ExternalToUtf.

**Tcl_WinUtfToTChar** and **Tcl_WinTCharToUtf** are Windows−only convenience functions for converting between UTF−8 and Windows strings. On Windows 95 (as with the Macintosh and Unix operating systems), all strings exchanged between Tcl and the operating system are "char" based. On Windows NT, some strings exchanged between Tcl and the operating system are "char" oriented while others are in Unicode. By convention, in Windows a TCHAR is a character in the ANSI code page on Windows 95 and a Unicode character on Windows NT.

If you planned to use the same "char" based interfaces on both Windows 95 and Windows NT, you could use Tcl_UtfToExternal and Tcl_ExternalToUtf (or their Tcl_DString equivalents) with an encoding of NULL (the current system encoding). On the other hand, if you planned to use the Unicode interface when running on Windows NT and the "char" interfaces when running on Windows 95, you would have to perform the following type of test over and over in your program (as represented in pseudo−code):

```c
if (running NT) {
    encoding <- Tcl_GetEncoding("unicode");
    nativeBuffer <- Tcl_UtfToExternal(encoding, utfBuffer);
    Tcl_FreeEncoding(encoding);
} else {
    nativeBuffer <- Tcl_UtfToExternal(NULL, utfBuffer);
}
```

**Tcl_WinUtfToTChar** and **Tcl_WinTCharToUtf** automatically handle this test and use the proper encoding based on the current operating system. Tcl_WinUtfToTChar returns a pointer to a TCHAR string, and Tcl_WinTCharToUtf expects a TCHAR string pointer as the src string. Otherwise, these functions behave identically to Tcl_UtfToExternalDString and Tcl_ExternalToUtfDString.

**Tcl_GetEncodingName** is roughly the inverse of Tcl_GetEncoding. Given an encoding, the return value is the name argument that was used to create the encoding. The string returned by Tcl_GetEncodingName is only guaranteed to persist until the encoding is deleted. The caller must not modify this string.
The `Tcl_SetSystemEncoding` function sets the default encoding that should be used whenever the user passes a NULL value for the `encoding` argument to any of the other encoding functions. If `name` is NULL, the system encoding is reset to the default system encoding, `binary`. If the name did not refer to any known or loadable encoding, `TCL_ERROR` is returned and an error message is left in `interp`. Otherwise, this procedure increments the reference count of the new system encoding, decrements the reference count of the old system encoding, and returns `TCL_OK`.

The `Tcl_GetEncodingNames` function sets the `interp` result to a list consisting of the names of all the encodings that are currently defined or can be dynamically loaded, searching the encoding path specified by `Tcl_SetDefaultEncodingDir`. This procedure does not ensure that the dynamically-loadable encoding files contain valid data, but merely that they exist.

The `Tcl_CreateEncoding` function defines a new encoding and registers the C procedures that are called back to convert between the encoding and UTF-8. Encodings created by `Tcl_CreateEncoding` are thereafter visible in the database used by `Tcl_GetEncoding`. Just as with the `Tcl_GetEncoding` procedure, the return value is a token that represents the encoding and can be used in subsequent calls to other encoding functions. `Tcl_CreateEncoding` returns an encoding with a reference count of 1. If an encoding with the specified `name` already exists, then its entry in the database is replaced with the new encoding; the token for the old encoding will remain valid and continue to behave as before, but users of the new token will now call the new encoding procedures.

The `typePtr` argument to `Tcl_CreateEncoding` contains information about the name of the encoding and the procedures that will be called to convert between this encoding and UTF-8. It is defined as follows:

```c
typedef struct Tcl_EncodingType {
    CONST char *encodingName;
    Tcl_EncodingConvertProc *toUtfProc;
    Tcl_EncodingConvertProc *fromUtfProc;
    Tcl_EncodingFreeProc *freeProc;
    ClientData clientData;
    int nullSize;
} Tcl_EncodingType;
```

The `encodingName` provides a string name for the encoding, by which it can be referred in other procedures such as `Tcl_GetEncoding`. The `toUtfProc` refers to a callback procedure to invoke to convert text from this encoding into UTF-8. The `fromUtfProc` refers to a callback procedure to invoke to convert text from UTF-8 into this encoding. The `freeProc` refers to a callback procedure to invoke when this encoding is deleted. The `freeProc` field may be NULL. The `clientData` contains an arbitrary one-word value passed to `toUtfProc`, `fromUtfProc`, and `freeProc` whenever they are called. Typically, this is a pointer to a data structure containing encoding-specific information that can be used by the callback procedures. For instance, two very similar encodings such as `ascii` and `macRoman` may use the same callback procedure, but use different values of `clientData` to control its behavior. The `nullSize` specifies the number of zero bytes that signify end-of-string in this encoding. It must be 1 (for single-byte or multi-byte encodings like ASCII or Shift-JIS) or 2 (for double-byte encodings like Unicode). Constant-sized encodings with 3 or more bytes per character (such as CNS11643) are not accepted.

The callback procedures `toUtfProc` and `fromUtfProc` should match the type `Tcl_EncodingConvertProc`.
typedef int Tcl_EncodingConvertProc(
    ClientData clientData,
    CONST char * src,
    int srcLen,
    int flags,
    Tcl_Encoding * statePtr,
    char * dst,
    int dstLen,
    int * srcReadPtr,
    int * dstWrotePtr,
    int * dstCharsPtr);

The `toUtfProc` and `fromUtfProc` procedures are called by the `Tcl.getExternalToUtf` or `Tcl.UtfToExternal` family of functions to perform the actual conversion. The `clientData` parameter to these procedures is the same as the `clientData` field specified to `Tcl.CreateEncoding` when the encoding was created. The remaining arguments to the callback procedures are the same as the arguments, documented at the top, to `Tcl.getExternalToUtf` or `Tcl.UtfToExternal`, with the following exceptions. If the `srcLen` argument to one of those high-level functions is negative, the value passed to the callback procedure will be the appropriate encoding-specific string length of `src`. If any of the `srcReadPtr`, `dstWrotePtr`, or `dstCharsPtr` arguments to one of the high-level functions is NULL, the corresponding value passed to the callback procedure will be a non-NULL location.

The callback procedure `freeProc`, if non-NULL, should match the type `Tcl_EncodingFreeProc`:

```c
typedef void Tcl_EncodingFreeProc(
    ClientData clientData);
```

This `freeProc` function is called when the encoding is deleted. The `clientData` parameter is the same as the `clientData` field specified to `Tcl.CreateEncoding` when the encoding was created.

`Tcl.GetDefaultEncodingDir` and `Tcl.SetDefaultEncodingDir` access and set the directory to use when locating the default encoding files. If this value is not NULL, the `TclpInitLibraryPath` routine appends the path to the head of the search path, and uses this path as the first place to look into when trying to locate the encoding file.

**ENCODING FILES**

Space would prohibit precompiling into Tcl every possible encoding algorithm, so many encodings are stored on disk as dynamically-loadable encoding files. This behavior also allows the user to create additional encoding files that can be loaded using the same mechanism. These encoding files contain information about the tables and/or escape sequences used to map between an external encoding and Unicode. The external encoding may consist of single-byte, multi-byte, or double-byte characters.

Each dynamically-loadable encoding is represented as a text file. The initial line of the file, beginning with a `"#"` symbol, is a comment that provides a human-readable description of the file. The next line identifies the type of encoding file. It can be one of the following letters:
A single-byte encoding, where one character is always one byte long in the encoding. An example is iso8859–1, used by many European languages.

A double-byte encoding, where one character is always two bytes long in the encoding. An example is big5, used for Chinese text.

A multi-byte encoding, where one character may be either one or two bytes long. Certain bytes are lead bytes, indicating that another byte must follow and that together the two bytes represent one character. Other bytes are not lead bytes and represent themselves. An example is shiftjis, used by many Japanese computers.

An escape-sequence encoding, specifying that certain sequences of bytes do not represent characters, but commands that describe how following bytes should be interpreted.

The rest of the lines in the file depend on the type.

Cases [1], [2], and [3] are collectively referred to as table-based encoding files. The lines in a table-based encoding file are in the same format as this example taken from the shiftjis encoding (this is not the complete file):

```
# Encoding file: shiftjis, multi-byte
M
003F 0 40
00
000000100020003000400050006000700080009000A000B000C000D000E000F
0100001100120013001400150016001700180019001A001B001C001D001E001F
02000201020003000400050006000700080009000A000B000C000D000E000F
0300030100020003000400050006000700080009000A000B000C000D000E000F
0400040100020003000400050006000700080009000A000B000C000D000E000F
0500050100020003000400050006000700080009000A000B000C000D000E000F
0600060100020003000400050006000700080009000A000B000C000D000E000F
0700070100020003000400050006000700080009000A000B000C000D000E000F
0800000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
81
```

ENCODING FILES

844
The third line of the file is three numbers. The first number is the fallback character (in base 16) to use when converting from UTF–8 to this encoding. The second number is a 1 if this file represents the encoding for a symbol font, or 0 otherwise. The last number (in base 10) is how many pages of data follow.

Subsequent lines in the example above are pages that describe how to map from the encoding into 2–byte Unicode. The first line in a page identifies the page number. Following it are 256 double–byte numbers, arranged as 16 rows of 16 numbers. Given a character in the encoding, the high byte of that character is used to select which page, and the low byte of that character is used as an index to select one of the double–byte numbers in that page – the value obtained being the corresponding Unicode character. By examination of the example above, one can see that the characters 0x7E and 0x8163 in shiftjis map to 203E and 2026 in Unicode, respectively.

Following the first page will be all the other pages, each in the same format as the first: one number identifying the page followed by 256 double–byte Unicode characters. If a character in the encoding maps to the Unicode character 0000, it means that the character doesn't actually exist. If all characters on a page would map to 0000, that page can be omitted.

Case [4] is the escape–sequence encoding file. The lines in an this type of file are in the same format as this example taken from the iso2022–jp encoding:

```
# Encoding file: iso2022–jp, escape–driven
E
init            {}
final           {}
iso8859–1       \x1b(B
jis0201         \x1b(J
jis0208         \x1b$B
jis0208         \x1b$A
jis0212         \x1b$D
gb2312          \x1b$A
ksc5601         \x1b$C
```

In the file, the first column represents an option and the second column is the associated value. init is a string to emit or expect before the first character is converted, while final is a string to emit or expect after the last character. All other options are names of table–based encodings; the associated value is the escape–sequence that marks that encoding. Tcl syntax is used for the values; in the above example, for instance, "{}" represents the empty string and "\x1b" represents character 27.

When Tcl_GetEncoding encounters an encoding name that has not been loaded, it attempts to load an encoding file called name.enc from the encoding subdirectory of each directory specified in the library path.
$tcl_libPath. If the encoding file exists, but is malformed, an error message will be left in interp.

KEYWORDS

utf, encoding, convert

Copyright © 1997−1998 Sun Microsystems, Inc.
Copyright © 1995−1997 Roger E. Critchlow Jr.
GetHostName

NAME

Tcl_GetHostName – get the name of the local host

SYNOPSIS

#include <tcl.h>
CONST char *
Tcl_GetHostName()

DESCRIPTION

Tcl_GetHostName is a utility procedure used by some of the Tcl commands. It returns a pointer to a string containing the name for the current machine, or an empty string if the name cannot be determined. The string is statically allocated, and the caller must not modify or free it.

KEYWORDS

hostname
GetIndex

NAME

Tcl_GetIndexFromObj, Tcl_GetIndexFromObjStruct – lookup string in table of keywords

SYNOPSIS

#include <tcl.h>

int Tcl_GetIndexFromObj(interp, objPtr, tablePtr, msg, flags, indexPtr)

int Tcl_GetIndexFromObjStruct(interp, objPtr, structTablePtr, offset, msg, flags, indexPtr)

ARGUMENTS

Tcl_Interp *interp (in)
    Interpreter to use for error reporting; if NULL, then no message is provided on errors.

Tcl_Obj *objPtr (in/out)
    The string value of this object is used to search through tablePtr. The internal representation is modified to hold the index of the matching table entry.

CONST char **tablePtr (in)
    An array of null–terminated strings. The end of the array is marked by a NULL string pointer.

CONST VOID *structTablePtr (in)
    An array of arbitrary type, typically some struct type. The first member of the structure must be a null–terminated string. The size of the structure is given by offset.

int offset (in)
    The offset to add to structTablePtr to get to the next entry. The end of the array is marked by a NULL string pointer.

CONST char *msg (in)
    Null–terminated string describing what is being looked up, such as option. This string is included in error messages.

int flags (in)
    OR–ed combination of bits providing additional information for operation. The only bit that is currently defined is TCL_EXACT.

int *indexPtr (out)
    The index of the string in tablePtr that matches the value of objPtr is returned here.

DESCRIPTION

This procedure provides an efficient way for looking up keywords, switch names, option names, and similar things where the value of an object must be one of a predefined set of values. ObjPtr is compared against each
of the strings in \texttt{tablePtr} to find a match. A match occurs if \texttt{objPtr}'s string value is identical to one of the strings in \texttt{tablePtr}, or if it is a unique abbreviation for exactly one of the strings in \texttt{tablePtr} and the \texttt{TCL\_EXACT} flag was not specified; in either case the index of the matching entry is stored at \texttt{*indexPtr} and \texttt{TCL\_OK} is returned.

If there is no matching entry, \texttt{TCL\_ERROR} is returned and an error message is left in \texttt{interp}'s result if \texttt{interp} isn't NULL. \texttt{Msg} is included in the error message to indicate what was being looked up. For example, if \texttt{msg} is \texttt{option} the error message will have a form like \texttt{bad option "firt": must be first, second, or third}.

If \texttt{Tcl\_GetIndexFromObj} completes successfully it modifies the internal representation of \texttt{objPtr} to hold the address of the table and the index of the matching entry. If \texttt{Tcl\_GetIndexFromObj} is invoked again with the same \texttt{objPtr} and \texttt{tablePtr} arguments (e.g. during a reinvocation of a Tcl command), it returns the matching index immediately without having to redo the lookup operation. Note: \texttt{Tcl\_GetIndexFromObj} assumes that the entries in \texttt{tablePtr} are static: they must not change between invocations. If the value of \texttt{objPtr} is the empty string, \texttt{Tcl\_GetIndexFromObj} will treat it as a non-matching value and return \texttt{TCL\_ERROR}.

\texttt{Tcl\_GetIndexFromObjStruct} works just like \texttt{Tcl\_GetIndexFromObj}, except that instead of treating \texttt{tablePtr} as an array of string pointers, it treats it as the first in a series of string ptrs that are spaced apart by \texttt{offset} bytes. This is particularly useful when processing things like \texttt{Tk\_ConfigurationSpec}, whose string keys are in the same place in each of several array elements.

\textbf{SEE ALSO}

\texttt{Tcl\_WrongNumArgs}

\textbf{KEYWORDS}

index, object, table lookup

\texttt{Copyright ' 1997 Sun Microsystems, Inc.}
\texttt{Copyright ' 1995-1997 Roger E. Critchlow Jr.}
GetInt

NAME

Tcl_GetInt, Tcl_GetDouble, Tcl_GetBoolean – convert from string to integer, double, or boolean

SYNOPSIS

#include <tcl.h>

int
Tcl_GetInt(interp, string, intPtr)
int
Tcl_GetDouble(interp, string, doublePtr)
int
Tcl_GetBoolean(interp, string, boolPtr)

ARGUMENTS

Tcl_Interp *interp (in)
Interpreter to use for error reporting.

CONST char *string (in)
Textual value to be converted.

int *intPtr (out)
Points to place to store integer value converted from string.

double *doublePtr (out)
Points to place to store double-precision floating-point value converted from string.

int *boolPtr (out)
Points to place to store boolean value (0 or 1) converted from string.

DESCRIPTION

These procedures convert from strings to integers or double-precision floating-point values or booleans (represented as 0- or 1-valued integers). Each of the procedures takes a string argument, converts it to an internal form of a particular type, and stores the converted value at the location indicated by the procedure's third argument. If all goes well, each of the procedures returns TCL_OK. If string doesn't have the proper syntax for the desired type then TCL_ERROR is returned, an error message is left in the interpreter's result, and nothing is stored at *intPtr or *doublePtr or *boolPtr.

Tcl_GetInt expects string to consist of a collection of integer digits, optionally signed and optionally preceded by white space. If the first two characters of string are ``0x'' then string is expected to be in hexadecimal form; otherwise, if the first character of string is ```0'' then string is expected to be in octal form; otherwise, string is expected to be in decimal form.
Tcl_GetDouble expects string to consist of a floating-point number, which is: white space; a sign; a sequence of digits; a decimal point; a sequence of digits; the letter `e'; and a signed decimal exponent. Any of the fields may be omitted, except that the digits either before or after the decimal point must be present and if the `e' is present then it must be followed by the exponent number.

Tcl_GetBoolean expects string to specify a boolean value. If string is any of 0, false, no, or off, then Tcl_GetBoolean stores a zero value at *boolPtr. If string is any of 1, true, yes, or on, then 1 is stored at *boolPtr. Any of these values may be abbreviated, and upper-case spellings are also acceptable.

KEYWORDS

boolean, conversion, double, floating-point, integer

Copyright ’ 1989-1993 The Regents of the University of California.  
Copyright ’ 1994-1996 Sun Microsystems, Inc.  
GetOpnFl

NAME

Tcl_GetOpenFile – Get a standard IO File * handle from a channel. (Unix only)

SYNOPSIS

#include <tcl.h>

int Tcl_GetOpenFile(interp, string, write, checkUsage, filePtr)

ARGUMENTS

Tcl_Interp *interp (in)
    Tcl interpreter from which file handle is to be obtained.
CONST char *string (in)
    String identifying channel, such as stdin or file4.
int write (in)
    Non–zero means the file will be used for writing, zero means it will be used for reading.
int checkUsage (in)
    If non–zero, then an error will be generated if the file wasn't opened for the access indicated by write.
ClientData *filePtr (out)
    Points to word in which to store pointer to FILE structure for the file given by string.

DESCRIPTION

Tcl_GetOpenFile takes as argument a file identifier of the form returned by the open command and returns at *filePtr a pointer to the FILE structure for the file. The write argument indicates whether the FILE pointer will be used for reading or writing. In some cases, such as a channel that connects to a pipeline of subprocesses, different FILE pointers will be returned for reading and writing. Tcl_GetOpenFile normally returns TCL_OK. If an error occurs in Tcl_GetOpenFile (e.g. string didn't make any sense or checkUsage was set and the file wasn't opened for the access specified by write) then TCL_ERROR is returned and the interpreter's result will contain an error message. In the current implementation checkUsage is ignored and consistency checks are always performed.

Note that this interface is only supported on the Unix platform.

KEYWORDS

channel, file handle, permissions, pipeline, read, write

Copyright © 1996–1997 Sun Microsystems, Inc.
GetStdChan

NAME

Tcl_GetStdChannel, Tcl_SetStdChannel – procedures for retrieving and replacing the standard channels

SYNOPSIS

#include <tcl.h>
Tcl_Channel
Tcl_GetStdChannel(type)
Tcl_SetStdChannel(channel, type)

ARGUMENTS

int type (in)
The identifier for the standard channel to retrieve or modify. Must be one of TCL_STDIN, TCL_STDOUT, or TCL_STDERR.

Tcl_Channel channel (in)
The channel to use as the new value for the specified standard channel.

DESCRIPTION

Tcl defines three special channels that are used by various I/O related commands if no other channels are specified. The standard input channel has a channel name of stdin and is used by read and reads. The standard output channel is named stdout and is used by puts. The standard error channel is named stderr and is used for reporting errors. In addition, the standard channels are inherited by any child processes created using exec or open in the absence of any other redirections.

The standard channels are actually aliases for other normal channels. The current channel associated with a standard channel can be retrieved by calling Tcl_GetStdChannel with one of TCL_STDIN, TCL_STDOUT, or TCL_STDERR as the type. The return value will be a valid channel, or NULL.

A new channel can be set for the standard channel specified by type by calling Tcl_SetStdChannel with a new channel or NULL in the channel argument. If the specified channel is closed by a later call to Tcl_Close, then the corresponding standard channel will automatically be set to NULL.

If Tcl_GetStdChannel is called before Tcl_SetStdChannel, Tcl will construct a new channel to wrap the appropriate platform–specific standard file handle. If Tcl_SetStdChannel is called before Tcl_GetStdChannel, then the default channel will not be created.

If one of the standard channels is set to NULL, either by calling Tcl_SetStdChannel with a NULL channel argument, or by calling Tcl_Close on the channel, then the next call to Tcl_CreateChannel will automatically set the standard channel with the newly created channel. If more than one standard channel is
NULL, then the standard channels will be assigned starting with standard input, followed by standard output, with standard error being last.

See `Tcl_StandardChannels` for a general treatise about standard channels and the behaviour of the Tcl library with regard to them.

**SEE ALSO**

`Tcl_Close`, `Tcl_CreateChannel`, `Tcl_Main`, `tclsh`

**KEYWORDS**

standard channel, standard input, standard output, standard error

---

*Copyright* © 1996 by Sun Microsystems, Inc.
GetTime

NAME

Tcl_GetTime – get date and time

SYNOPSIS

#include <tcl.h>
Tcl_GetTime( timePtr )

ARGUMENTS

Tcl_Time * timePtr (out)
    Points to memory in which to store the date and time information.

DESCRIPTION

The Tcl_GetTime function retrieves the current time as a Tcl_Time structure in memory the caller provides. This structure has the following definition:

typedef struct Tcl_Time {
    long sec;
    long usec;
} Tcl_Time;

On return, the sec member of the structure is filled in with the number of seconds that have elapsed since the epoch: the epoch is the point in time of 00:00 UTC, 1 January 1970. This number does not count leap seconds – an interval of one day advances it by 86400 seconds regardless of whether a leap second has been inserted.

The usec member of the structure is filled in with the number of microseconds that have elapsed since the start of the second designated by sec. The Tcl library makes every effort to keep this number as precise as possible, subject to the limitations of the computer system. On multiprocessor variants of Windows, this number may be limited to the 10– or 20–ms granularity of the system clock. (On single–processor Windows systems, the usec field is derived from a performance counter and is highly precise.)

SEE ALSO

clock

KEYWORDS

date, time
GetVersion

NAME

Tcl_GetVersion – get the version of the library at runtime

SYNOPSIS

#include <tcl.h>
Tcl_GetVersion(major, minor, patchLevel, type)

ARGUMENTS

int *major (out)
   Major version number of the Tcl library.
int *minor (out)
   Minor version number of the Tcl library.
int *patchLevel (out)
   The patch level of the Tcl library (or alpha or beta number).
Tcl_ReleaseType *type (out)
   The type of release, also indicates the type of patch level. Can be one of TCL_ALPHA_RELEASE,
   TCL_BETA_RELEASE, or TCL_FINAL_RELEASE.

DESCRIPTION

Tcl_GetVersion should be used to query the version number of the Tcl library at runtime. This is useful when
using a dynamically loaded Tcl library or when writing a stubs–aware extension. For instance, if you write an
extension that is linked against the Tcl stubs library, it could be loaded into a program linked to an older
version of Tcl than you expected. Use Tcl_GetVersion to verify that fact, and possibly to change the behavior
of your extension.

Tcl_GetVersion accepts NULL for any of the arguments. For instance if you do not care about the patchLevel
of the library, pass a NULL for the patchLevel argument.

KEYWORDS

version, patchlevel, major, minor, alpha, beta, release
Hash

NAME

Tcl_InitHashTable, Tcl_InitCustomHashTable, Tcl_InitObjHashTable,
Tcl_DeleteHashTable, Tcl_CreateHashEntry, Tcl_DeleteHashEntry, Tcl_FindHashEntry,
Tcl_GetHashValue, Tcl_SetHashValue, Tcl_GetHashKey, Tcl_FirstHashEntry,
Tcl_NextHashEntry, Tcl_HashStats – procedures to manage hash tables

SYNOPSIS

ARGUMENTS

DESCRIPTION

TCL STRING KEYS
TCL ONE WORD KEYS
TCL CUSTOM TYPE KEYS
TCL CUSTOM PTR KEYS
other

THE TCL_HASHKEYTYPE STRUCTURE
TCL HASH KEY RANDOMIZE HASH

KEYWORDS

NAME

Tcl_InitHashTable, Tcl_InitCustomHashTable, Tcl_InitObjHashTable, Tcl_DeleteHashTable,
Tcl_CreateHashEntry, Tcl_DeleteHashEntry, Tcl_FindHashEntry, Tcl_GetHashValue, Tcl_SetHashValue,
Tcl_GetHashKey, Tcl_FirstHashEntry, Tcl_NextHashEntry, Tcl_HashStats – procedures to manage hash tables

SYNOPSIS

#include <tcl.h>

Tcl_InitHashTable(tablePtr, keyType)
Tcl_InitCustomHashTable(tablePtr, keyType, typePtr)
Tcl_InitObjHashTable(tablePtr)
Tcl_DeleteHashTable(tablePtr)
Tcl_CreateHashEntry(tablePtr, key, newPtr)
Tcl_DeleteHashEntry(entryPtr)
Tcl_FirstHashEntry *
Tcl_FindHashEntry(tablePtr, key)
Tcl_GetHashValue(entryPtr)
ClientData
Tcl_SetHashValue(entryPtr, value)
char *
**Tcl_GetHashKey**(tablePtr, entryPtr)
Tcl_HashEntry *

**Tcl_FirstHashEntry**(tablePtr, searchPtr)
Tcl_HashEntry *

**Tcl_NextHashEntry**(searchPtr)
CONST char *

**Tcl_HashStats**(tablePtr)

**ARGUMENTS**

*Tcl_HashTable* *tablePtr* *(in)*
Address of hash table structure (for all procedures but *Tcl_InitHashTable*, this must have been initialized by previous call to *Tcl_InitHashTable*).

*int* keyType *(in)*
Kind of keys to use for new hash table. Must be either TCL_STRING_KEYS, TCL_ONE_WORD_KEYS, TCL_CUSTOM_TYPE_KEYS, TCL_CUSTOM_PTR_KEYS, or an integer value greater than 1.

*Tcl_HashKeyType* *typePtr* *(in)*
Address of structure which defines the behaviour of the hash table.

*CONST char* *key* *(in)*
Key to use for probe into table. Exact form depends on keyType used to create table.

*int* *newPtr* *(out)*
The word at *newPtr* is set to 1 if a new entry was created and 0 if there was already an entry for *key*.

*Tcl_HashEntry* *entryPtr* *(in)*
Pointer to hash table entry.

*ClientData* value *(in)*
New value to assign to hash table entry. Need not have type ClientData, but must fit in same space as ClientData.

*Tcl_HashSearch* *searchPtr* *(in)*
Pointer to record to use to keep track of progress in enumerating all the entries in a hash table.

**DESCRIPTION**

A hash table consists of zero or more entries, each consisting of a key and a value. Given the key for an entry, the hashing routines can very quickly locate the entry, and hence its value. There may be at most one entry in a hash table with a particular key, but many entries may have the same value. Keys can take one of four forms: strings, one-word values, integer arrays, or custom keys defined by a Tcl_HashKeyType structure (See section **THE TCL_HASHKEYTYPE STRUCTURE** below). All of the keys in a given table have the same form, which is specified when the table is initialized.

The value of a hash table entry can be anything that fits in the same space as a ``char *`` pointer. Values for hash table entries are managed entirely by clients, not by the hash module itself. Typically each entry's value is a pointer to a data structure managed by client code.
Hash tables grow gracefully as the number of entries increases, so that there are always less than three entries per hash bucket, on average. This allows for fast lookups regardless of the number of entries in a table.

The core provides three functions for the initialization of hash tables, Tcl_InitHashTable, Tcl_InitObjHashTable and Tcl_InitCustomHashTable.

**Tcl_InitHashTable** initializes a structure that describes a new hash table. The space for the structure is provided by the caller, not by the hash module. The value of *keyType* indicates what kinds of keys will be used for all entries in the table. All of the key types described later are allowed, with the exception of **TCL_CUSTOM_TYPE_KEYS** and **TCL_CUSTOM_PTR_KEYS**.

**Tcl_InitObjHashTable** is a wrapper around **Tcl_InitCustomHashTable** and initializes a hash table whose keys are Tcl_Obj *.

**Tcl_InitCustomHashTable** initializes a structure that describes a new hash table. The space for the structure is provided by the caller, not by the hash module. The value of *keyType* indicates what kinds of keys will be used for all entries in the table. **KeyType** must have one of the following values:

**TCL_STRING_KEYS**
Keys are null-terminated strings. They are passed to hashing routines using the address of the first character of the string.

**TCL_ONE_WORD_KEYS**
Keys are single-word values; they are passed to hashing routines and stored in hash table entries as `char *` values. The pointer value is the key; it need not (and usually doesn't) actually point to a string.

**TCL_CUSTOM_TYPE_KEYS**
Keys are of arbitrary type, and are stored in the entry. Hashing and comparison is determined by *typePtr*. The Tcl_HashKeyType structure is described in the section **THE TCL_HASHKEYTYPE STRUCTURE** below.

**TCL_CUSTOM_PTR_KEYS**
Keys are pointers to an arbitrary type, and are stored in the entry. Hashing and comparison is determined by *typePtr*. The Tcl_HashKeyType structure is described in the section **THE TCL_HASHKEYTYPE STRUCTURE** below.

**other**
If *keyType* is not one of the above, then it must be an integer value greater than 1. In this case the keys will be arrays of `int` values, where *keyType* gives the number of ints in each key. This allows structures to be used as keys. All keys must have the same size. Array keys are passed into hashing functions using the address of the first int in the array.

**Tcl_DeleteHashTable** deletes all of the entries in a hash table and frees up the memory associated with the table's bucket array and entries. It does not free the actual table structure (pointed to by *tablePtr*), since that memory is assumed to be managed by the client. **Tcl_DeleteHashTable** also does not free or otherwise manipulate the values of the hash table entries. If the entry values point to dynamically-allocated memory, then it is the client's responsibility to free these structures before deleting the table.
**Tcl_CreateHashEntry** locates the entry corresponding to a particular key, creating a new entry in the table if there wasn't already one with the given key. If an entry already existed with the given key then *newPtr* is set to zero. If a new entry was created, then *newPtr* is set to a non-zero value and the value of the new entry will be set to zero. The return value from **Tcl_CreateHashEntry** is a pointer to the entry, which may be used to retrieve and modify the entry's value or to delete the entry from the table.

**Tcl_DeleteHashEntry** will remove an existing entry from a table. The memory associated with the entry itself will be freed, but the client is responsible for any cleanup associated with the entry's value, such as freeing a structure that it points to.

**Tcl_FindHashEntry** is similar to **Tcl_CreateHashEntry** except that it doesn't create a new entry if the key doesn't exist; instead, it returns NULL as result.

**Tcl_GetHashValue** and **Tcl_SetHashValue** are used to read and write an entry's value, respectively. Values are stored and retrieved as type ``ClientData'', which is large enough to hold a pointer value. On almost all machines this is large enough to hold an integer value too.

**Tcl_GetHashKey** returns the key for a given hash table entry, either as a pointer to a string, a one-word (``char *'') key, or as a pointer to the first word of an array of integers, depending on the keyType used to create a hash table. In all cases **Tcl_GetHashKey** returns a result with type ``char *''. When the key is a string or array, the result of **Tcl_GetHashKey** points to information in the table entry; this information will remain valid until the entry is deleted or its table is deleted.

**Tcl_FirstHashEntry** and **Tcl_NextHashEntry** may be used to scan all of the entries in a hash table. A structure of type ````Tcl_HashSearch'', provided by the client, is used to keep track of progress through the table. **Tcl_FirstHashEntry** initializes the search record and returns the first entry in the table (or NULL if the table is empty). Each subsequent call to **Tcl_NextHashEntry** returns the next entry in the table or NULL if the end of the table has been reached. A call to **Tcl_FirstHashEntry** followed by calls to **Tcl_NextHashEntry** will return each of the entries in the table exactly once, in an arbitrary order. It is unadvisable to modify the structure of the table, e.g. by creating or deleting entries, while the search is in progress.

**Tcl_HashStats** returns a dynamically-allocated string with overall information about a hash table, such as the number of entries it contains, the number of buckets in its hash array, and the utilization of the buckets. It is the caller's responsibility to free the result string by passing it to ckfree.

The header file tcl.h defines the actual data structures used to implement hash tables. This is necessary so that clients can allocate Tcl_HashTable structures and so that macros can be used to read and write the values of entries. However, users of the hashing routines should never refer directly to any of the fields of any of the hash-related data structures; use the procedures and macros defined here.

**THE TCL_HASHKEYTYPE STRUCTURE**

Extension writers can define new hash key types by defining four procedures, initializing a Tcl_HashKeyType structure to describe the type, and calling **Tcl_InitCustomHashTable**. The **Tcl_HashKeyType** structure is
defined as follows:

typedef struct Tcl_HashKeyType {
    int version;
    int flags;
    Tcl_HashKeyProc *hashKeyProc;
    Tcl_CompareHashKeysProc *compareKeysProc;
    Tcl_AllocHashEntryProc *allocEntryProc;
    Tcl_FreeHashEntryProc *freeEntryProc;
} Tcl_HashKeyType;

The version member is the version of the table. If this structure is extended in future then the version can be used to distinguish between different structures. It should be set to TCL_HASH_KEY_TYPE_VERSION.

The flags member is one or more of the following values OR'ed together:

**TCL_HASH_KEY_RANDOMIZE_HASH**
There are some things, pointers for example which don't hash well because they do not use the lower bits. If this flag is set then the hash table will attempt to rectify this by randomising the bits and then using the upper N bits as the index into the table.

The hashKeyProc member contains the address of a function called to calculate a hash value for the key.

typedef unsigned int (Tcl_HashKeyProc) (Tcl_HashTable *tablePtr, VOID *keyPtr);

If this is NULL then keyPtr is used and TCL_HASH_KEY_RANDOMIZE_HASH is assumed.

The compareKeysProc member contains the address of a function called to compare two keys.

typedef int (Tcl_CompareHashKeysProc) (VOID *keyPtr, Tcl_HashEntry *hPtr);

If this is NULL then the keyPtr pointers are compared. If the keys don't match then the function returns 0, otherwise it returns 1.

The allocEntryProc member contains the address of a function called to allocate space for an entry and initialise the key.

typedef Tcl_HashEntry *(Tcl_AllocHashEntryProc) (Tcl_HashTable *tablePtr, VOID *keyPtr);

If this is NULL then Tcl_Alloc is used to allocate enough space for a Tcl_HashEntry and the key pointer is assigned to key.oneWordValue. String keys and array keys use this function to allocate enough space for the entry and the key in one block, rather than doing it in two blocks. This saves space for a pointer to the key from the entry and another memory allocation. Tcl_Obj * keys use this function to allocate enough space for an entry and increment the reference count on the object.
The *freeEntryProc* member contains the address of a function called to free space for an entry.

```c
typedef void (Tcl_FreeHashEntryProc) (Tcl_HashEntry *hPtr);
```

If this is NULL then *Tcl_Free* is used to free the space for the entry. *Tcl_Obj* *keys* use this function to decrement the reference count on the object.

**KEYWORDS**

hash table, key, lookup, search, value
Init

NAME

Tcl_Init – find and source initialization script

SYNOPSIS

#include <tcl.h>

int
Tcl_Init(interp)

ARGUMENTS

Tcl_Interp *interp (in)

Interpreter to initialize.

DESCRIPTION

Tcl_Init is a helper procedure that finds and source's the init.tcl script, which should exist somewhere on the Tcl library path. On Macintosh systems, it additionally checks for an Init resource and sources the contents of that resource if init.tcl cannot be found.

Tcl_Init is typically called from Tcl_AppInit procedures.

SEE ALSO

Tcl_AppInit, Tcl_Main

KEYWORDS

application, initialization, interpreter

Copyright ' 1995-1997 Roger E. Critchlow Jr.
InitStubs

NAME

Tcl_InitStubs − initialize the Tcl stubs mechanism

SYNOPSIS

ARGUMENTS

INTRODUCTION

DESCRIPTION

SEE ALSO

KEYWORDS

NAME

Tcl_InitStubs − initialize the Tcl stubs mechanism

SYNOPSIS

#include <tcl.h>

CONST char *
Tcl_InitStubs(interp, version, exact)

ARGUMENTS

Tcl_Interp *interp (in)
    Tcl interpreter handle.

CONST char *version (in)
    A version string consisting of one or more decimal numbers separated by dots.

int exact (in)
    Non−zero means that only the particular version specified by version is acceptable. Zero means that versions newer than version are also acceptable as long as they have the same major version number as version.

INTRODUCTION

The Tcl stubs mechanism defines a way to dynamically bind extensions to a particular Tcl implementation at run time. This provides two significant benefits to Tcl users:
1) Extensions that use the stubs mechanism can be loaded into multiple versions of Tcl without being recompiled or relinked.

2) Extensions that use the stubs mechanism can be dynamically loaded into statically-linked Tcl applications.

The stubs mechanism accomplishes this by exporting function tables that define an interface to the Tcl API. The extension then accesses the Tcl API through offsets into the function table, so there are no direct references to any of the Tcl library's symbols. This redirection is transparent to the extension, so an extension writer can continue to use all public Tcl functions as documented.

The stubs mechanism requires no changes to applications incorporating Tcl interpreters. Only developers creating C-based Tcl extensions need to take steps to use the stubs mechanism with their extensions.

Enabling the stubs mechanism for an extension requires the following steps:

1) Call `Tcl_InitStubs` in the extension before calling any other Tcl functions.

2) Define the USE_TCL_STUBS symbol. Typically, you would include the `-DUSE_TCL_STUBS` flag when compiling the extension.

3) Link the extension with the Tcl stubs library instead of the standard Tcl library. On Unix platforms, the library name is `libtclstub8.1.a`; on Windows platforms, the library name is `tclstub81.lib`.

If the extension also requires the Tk API, it must also call `Tk_InitStubs` to initialize the Tk stubs interface and link with the Tk stubs libraries. See the `Tk_InitStubs` page for more information.

**DESCRIPTION**

`Tcl_InitStubs` attempts to initialize the stub table pointers and ensure that the correct version of Tcl is loaded. In addition to an interpreter handle, it accepts as arguments a version number and a Boolean flag indicating whether the extension requires an exact version match or not. If `exact` is 0, then the extension is indicating that newer versions of Tcl are acceptable as long as they have the same major version number as `version`; non-zero means that only the specified `version` is acceptable. `Tcl_InitStubs` returns a string containing the actual version of Tcl satisfying the request, or NULL if the Tcl version is not acceptable, does not support stubs, or any other error condition occurred.

**SEE ALSO**

`Tk_InitStubs`
KEYWORDS

stubs
**NAME**

Tcl_NewIntObj, Tcl_NewLongObj, Tcl_NewWideIntObj, Tcl_SetIntObj, Tcl_SetLongObj, 
Tcl_SetWideIntObj, Tcl_GetIntFromObj, Tcl_GetLongFromObj, Tcl_GetWideIntFromObj – manipulate Tcl 
objects as integers and wide integers

**SYNOPSIS**

```c
#include <tcl.h>

Tcl_Obj *
Tcl_NewIntObj(intValue)
Tcl_Obj *
Tcl_NewLongObj(longValue)
Tcl_Obj *
Tcl_NewWideIntObj(wideValue)
Tcl_SetIntObj(objPtr, intValue)
Tcl_SetLongObj(objPtr, longValue)
Tcl_SetWideIntObj(objPtr, wideValue)

int
Tcl_GetIntFromObj(interp, objPtr, intPtr)
int
Tcl_GetLongFromObj(interp, objPtr, longPtr)
int
Tcl_GetWideIntFromObj(interp, objPtr, widePtr)
```

**ARGUMENTS**

int **intValue** *(in)*

Integer value used to initialize or set an integer object.

long **longValue** *(in)*

Long integer value used to initialize or set an integer object.

Tcl_WideInt **wideValue** *(in)*

Wide integer value (minimum 64−bits wide where supported by the compiler) used to initialize or set 
a wide integer object.

Tcl_Obj **objPtr** *(in/out)*

For **Tcl_SetIntObj**, **Tcl_SetLongObj**, and **Tcl_SetWideIntObj**, this points to the object to be 
converted to integer type. For **Tcl_GetIntFromObj**, **Tcl_GetLongFromObj**, and 
**Tcl_GetWideIntFromObj**, this refers to the object from which to get an integer or long integer 
value; if **objPtr** does not already point to an integer object (or a wide integer object in the case of 
**Tcl_SetWideIntObj** and **Tcl_GetWideIntFromObj**), an attempt will be made to convert it to one.

Tcl_Interp **interp** *(in/out)*

If an error occurs during conversion, an error message is left in the interpreter's result object unless
interp is NULL.

int *intPtr (out)
Points to place to store the integer value obtained by Tcl_GetIntFromObj from objPtr.

long *longPtr (out)
Points to place to store the long integer value obtained by Tcl_GetLongFromObj from objPtr.

Tcl_WideInt *widePtr (out)
Points to place to store the wide integer value obtained by Tcl_GetWideIntFromObj from objPtr.

DESCRIPTION

These procedures are used to create, modify, and read integer and wide integer Tcl objects from C code. Tcl_NewIntObj, Tcl_NewLongObj, Tcl_SetIntObj, and Tcl_SetLongObj create a new object of integer type or modify an existing object to have integer type, and Tcl_NewWideIntObj and Tcl_SetWideObj create a new object of wide integer type or modify an existing object to have wide integer type.

Tcl_NewIntObj and Tcl_SetIntObj set the object to have the integer value given by intValue, Tcl_NewLongObj and Tcl_SetLongObj set the object to have the long integer value given by longValue, and Tcl_NewWideIntObj and Tcl_SetWideObj set the object to have the wide integer value given by wideValue. Tcl_NewIntObj, Tcl_NewLongObj and Tcl_NewWideIntObj return a pointer to a newly created object with reference count zero. These procedures set the object's type to be integer and assign the integer value to the object's internal representation longValue or wideValue member (as appropriate). Tcl_SetIntObj, Tcl_SetLongObj and Tcl_SetWideObj invalidate any old string representation and, if the object is not already an integer object, free any old internal representation.

Tcl_GetIntFromObj and Tcl_GetLongFromObj attempt to return an integer value from the Tcl object objPtr, and Tcl_GetWideIntFromObj attempts to return a wide integer value from the Tcl object objPtr. If the object is not already an integer object, or a wide integer object in the case of Tcl_GetWideIntFromObj they will attempt to convert it to one. If an error occurs during conversion, they return TCL_ERROR and leave an error message in the interpreter's result object unless interp is NULL. Also, if the long integer held in the object's internal representation longValue member can not be represented in a (non-long) integer, Tcl_GetIntFromObj returns TCL_ERROR and leaves an error message in the interpreter's result object unless interp is NULL. Otherwise, all three procedures return TCL_OK and store the integer, long integer value or wide integer in the address given by intPtr, longPtr and widePtr respectively. If the object is not already an integer or wide integer object, the conversion will free any old internal representation.

SEE ALSO

Tcl_NewObj, Tcl_DecrRefCount, Tcl_IncrRefCount, Tcl_GetObjResult

KEYWORDS

copyright '1996-1997 Sun Microsystems, Inc.
Interp

NAME

Tcl_Interp – client−visible fields of interpreter structures

SYNOPSIS

```c
#include <tcl.h>
typedef struct {
    char *result;
    Tcl_FreeProc *freeProc;
    int errorLine;
} Tcl_Interp;
```

typedef void Tcl_FreeProc(char *blockPtr);

DESCRIPTION

The Tcl_CreateInterp procedure returns a pointer to a Tcl_Interp structure. This pointer is then passed into other Tcl procedures to process commands in the interpreter and perform other operations on the interpreter. Interpreter structures contain many many fields that are used by Tcl, but only three that may be accessed by clients: result, freeProc, and errorLine.

The result and freeProc fields are used to return results or error messages from commands. This information is returned by command procedures back to Tcl_Eval, and by Tcl_Eval back to its callers. The result field points to the string that represents the result or error message, and the freeProc field tells how to dispose of the storage for the string when it isn't needed anymore. The easiest way for command procedures to manipulate these fields is to call procedures like Tcl_SetResult or Tcl_AppendResult; they will hide all the details of managing the fields. The description below is for those procedures that manipulate the fields directly.

Whenever a command procedure returns, it must ensure that the result field of its interpreter points to the string being returned by the command. The result field must always point to a valid string. If a command wishes to return no result then interp−>result should point to an empty string. Normally, results are assumed to be statically allocated, which means that the contents will not change before the next time Tcl_Eval is called or some other command procedure is invoked. In this case, the freeProc field must be zero. Alternatively, a command procedure may dynamically allocate its return value (e.g. using Tcl_Alloc) and store a pointer to it in interp−>result. In this case, the command procedure must also set interp−>freeProc to the address of a procedure that can free the value, or TCL_DYNAMIC if the storage was allocated directly by Tcl or by a call to Tcl_Alloc. If interp−>freeProc is non−zero, then Tcl will call freeProc to free the space pointed to by interp−>result before it invokes the next command. If a client procedure overwrites interp−>result when interp−>freeProc is non−zero, then it is responsible for calling freeProc to free the old interp−>result (the Tcl_FreeResult macro should be used for this purpose).
FreeProc should have arguments and result that match the Tcl_FreeProc declaration above: it receives a single argument which is a pointer to the result value to free. In most applications TCL_DYNAMIC is the only non–zero value ever used for freeProc. However, an application may store a different procedure address in freeProc in order to use an alternate memory allocator or in order to do other cleanup when the result memory is freed.

As part of processing each command, Tcl_Eval initializes interp->result and interp->freeProc just before calling the command procedure for the command. The freeProc field will be initialized to zero, and interp->result will point to an empty string. Commands that do not return any value can simply leave the fields alone. Furthermore, the empty string pointed to by result is actually part of an array of TCL_RESULT_SIZE characters (approximately 200). If a command wishes to return a short string, it can simply copy it to the area pointed to by interp->result. Or, it can use the sprintf procedure to generate a short result string at the location pointed to by interp->result.

It is a general convention in Tcl–based applications that the result of an interpreter is normally in the initialized state described in the previous paragraph. Procedures that manipulate an interpreter's result (e.g. by returning an error) will generally assume that the result has been initialized when the procedure is called. If such a procedure is to be called after the result has been changed, then Tcl_SetResult should be called first to reset the result to its initialized state. The direct use of interp->result is strongly deprecated (see Tcl_SetResult).

The errorLine field is valid only after Tcl_Eval returns a TCL_ERROR return code. In this situation the errorLine field identifies the line number of the command being executed when the error occurred. The line numbers are relative to the command being executed: 1 means the first line of the command passed to Tcl_Eval, 2 means the second line, and so on. The errorLine field is typically used in conjunction with Tcl_AddErrorInfo to report information about where an error occurred. ErrorLine should not normally be modified except by Tcl_Eval.

KEYWORDS
free, initialized, interpreter, malloc, result

Copyright ' 1989–1993 The Regents of the University of California.
Copyright ' 1994–1996 Sun Microsystems, Inc.
LinkVar

NAME

Tcl_LinkVar, Tcl_UnlinkVar, Tcl_UpdateLinkedVar − link Tcl variable to C variable

SYNOPSIS

ARGUMENTS

DESCRIPTION

TCL_LINK_INT
TCL_LINK_DOUBLE
TCL_LINK_WIDE_INT
TCL_LINK_BOOLEAN
TCL_LINK_STRING

KEYWORDS

NAME

Tcl_LinkVar, Tcl_UnlinkVar, Tcl_UpdateLinkedVar − link Tcl variable to C variable

SYNOPSIS

#include <tcl.h>

int
Tcl_LinkVar(interp, varName, addr, type)
Tcl_UnlinkVar(interp, varName)
Tcl_UpdateLinkedVar(interp, varName)

ARGUMENTS

Tcl_Interp *interp (in)
    Interpreter that contains varName. Also used by Tcl_LinkVar to return error messages.

CONST char *varName (in)
    Name of global variable.

char *addr (in)
    Address of C variable that is to be linked to varName.

int type (in)
    Type of C variable. Must be one of TCL_LINK_INT, TCL_LINK_DOUBLE,
    TCL_LINK_WIDE_INT, TCL_LINK_BOOLEAN, or TCL_LINK_STRING, optionally OR'ed with
    TCL_LINK_READ_ONLY to make Tcl variable read-only.
**DESCRIPTION**

_Tcl_LinkVar_ uses variable traces to keep the Tcl variable named by `varName` in sync with the C variable at the address given by `addr`. Whenever the Tcl variable is read the value of the C variable will be returned, and whenever the Tcl variable is written the C variable will be updated to have the same value. _Tcl_LinkVar_ normally returns TCL_OK; if an error occurs while setting up the link (e.g. because `varName` is the name of array) then TCL_ERROR is returned and the interpreter's result contains an error message.

The _type_ argument specifies the type of the C variable, and must have one of the following values, optionally OR'ed with TCL_LINK_READ_ONLY:

- **TCL_LINK_INT**
  - The C variable is of type int. Any value written into the Tcl variable must have a proper integer form acceptable to _Tcl_GetIntFromObj_; attempts to write non–integer values into `varName` will be rejected with Tcl errors.

- **TCL_LINK_DOUBLE**
  - The C variable is of type double. Any value written into the Tcl variable must have a proper real form acceptable to _Tcl_GetDoubleFromObj_; attempts to write non−real values into `varName` will be rejected with Tcl errors.

- **TCL_LINK_WIDE_INT**
  - The C variable is of type _Tcl_WideInt_ (which is an integer type at least 64–bits wide on all platforms that can support it.) Any value written into the Tcl variable must have a proper integer form acceptable to _Tcl_GetWideIntFromObj_; attempts to write non–integer values into `varName` will be rejected with Tcl errors.

- **TCL_LINK_BOOLEAN**
  - The C variable is of type int. If its value is zero then it will read from Tcl as ``0''; otherwise it will read from Tcl as ``1''. Whenever `varName` is modified, the C variable will be set to a 0 or 1 value. Any value written into the Tcl variable must have a proper boolean form acceptable to _Tcl_GetBooleanFromObj_; attempts to write non–boolean values into `varName` will be rejected with Tcl errors.

- **TCL_LINK_STRING**
  - The C variable is of type char *. If its value is not NULL then it must be a pointer to a string allocated with _Tcl_Alloc_ or _ckalloc_. Whenever the Tcl variable is modified the current C string will be freed and new memory will be allocated to hold a copy of the variable's new value. If the C variable contains a NULL pointer then the Tcl variable will read as ``NULL''.

If the TCL_LINK_READ_ONLY flag is present in _type_ then the variable will be read–only from Tcl, so that its value can only be changed by modifying the C variable. Attempts to write the variable from Tcl will be rejected with errors.

_Tcl_UnlinkVar_ removes the link previously set up for the variable given by `varName`. If there does not exist a link for `varName` then the procedure has no effect.

_Tcl_UpdateLinkedVar_ may be invoked after the C variable has changed to force the Tcl variable to be updated immediately. In many cases this procedure is not needed, since any attempt to read the Tcl variable
will return the latest value of the C variable. However, if a trace has been set on the Tcl variable (such as a Tk widget that wishes to display the value of the variable), the trace will not trigger when the C variable has changed. `Tcl_UpdateLinkedVar` ensures that any traces on the Tcl variable are invoked.

**KEYWORDS**

boolean, integer, link, read-only, real, string, traces, variable

---

Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
Copyright © 1995-1997 Roger E. Critchlow Jr.
**ListObj**

**NAME**

Tcl_ListObjAppendList, Tcl_ListObjAppendElement, Tcl_NewListObj, Tcl_SetListObj, Tcl_ListObjGetElements, Tcl_ListObjLength, Tcl_ListObjIndex, Tcl_ListObjReplace — manipulate Tcl objects as lists

**SYNOPSIS**

```c
#include <tcl.h>
int  Tcl_ListObjAppendList(interp, listPtr, elemListPtr)
int  Tcl_ListObjAppendElement(interp, listPtr, objPtr)
Tcl_Obj *  Tcl_NewListObj(objc, objv)
Tcl_SetListObj(objPtr, objc, objv)
int  Tcl_ListObjGetElements(interp, listPtr, objcPtr, objvPtr)
int  Tcl_ListObjLength(interp, listPtr, intptr)
int  Tcl_ListObjIndex(interp, listPtr, index, objPtrPtr)
int  Tcl_ListObjReplace(interp, listPtr, first, count, objc, objv)
```

**ARGUMENTS**

*interp (in)*

If an error occurs while converting an object to be a list object, an error message is left in the interpreter’s result object unless interp is NULL.

*listPtr (in/out)*

Points to the list object to be manipulated. If listPtr does not already point to a list object, an attempt will be made to convert it to one.

*elemListPtr (in/out)*

For Tcl_ListObjAppendList, this points to a list object containing elements to be appended onto listPtr. Each element of *elemListPtr will become a new element of listPtr. If *elemListPtr is not NULL and does not already point to a list object, an attempt will be made to convert it to one.

*objPtr (in)*

For Tcl_ListObjAppendElement, points to the Tcl object that will be appended to listPtr. For Tcl_SetListObj, this points to the Tcl object that will be converted to a list object containing the objc elements of the array referenced by objv.

*objcPtr (in)*


Points to location where Tcl_ListObjGetElements stores the number of element objects in listPtr.

_Tcl_Obj **objvPtr (out)

A location where Tcl_ListObjGetElements stores a pointer to an array of pointers to the element objects of listPtr.

int objc (in)
The number of Tcl objects that Tcl_NewListObj will insert into a new list object, and Tcl_ListObjReplace will insert into listPtr. For Tcl_SetListObj, the number of Tcl objects to insert into objPtr.

_Tcl_Obj *CONST objv[] (in)
An array of pointers to objects. Tcl_NewListObj will insert these objects into a new list object and Tcl_ListObjReplace will insert them into an existing listPtr. Each object will become a separate list element.

int *intPtr (out)
Points to location where Tcl_ListObjLength stores the length of the list.

int index (in)
Index of the list element that Tcl_ListObjIndex is to return. The first element has index 0.

_Tcl_Obj **objPtrPtr (out)
Points to place where Tcl_ListObjIndex is to store a pointer to the resulting list element object.

int first (in)
Index of the starting list element that Tcl_ListObjReplace is to replace. The list's first element has index 0.

int count (in)
The number of elements that Tcl_ListObjReplace is to replace.

DESCRIPTION

Tcl list objects have an internal representation that supports the efficient indexing and appending. The procedures described in this man page are used to create, modify, index, and append to Tcl list objects from C code.

_Tcl_ListObjAppendList and Tcl_ListObjAppendElement both add one or more objects to the end of the list object referenced by listPtr. Tcl_ListObjAppendList appends each element of the list object referenced by elemListPtr while Tcl_ListObjAppendElement appends the single object referenced by objPtr. Both procedures will convert the object referenced by listPtr to a list object if necessary. If an error occurs during conversion, both procedures return TCL_ERROR and leave an error message in the interpreter's result object if interp is not NULL. Similarly, if elemListPtr does not already refer to a list object, Tcl_ListObjAppendList will attempt to convert it to one and if an error occurs during conversion, will return TCL_ERROR and leave an error message in the interpreter's result object if interp is not NULL. Both procedures invalidate any old string representation of listPtr and, if it was converted to a list object, free any old internal representation. Similarly, Tcl_ListObjAppendList frees any old internal representation of elemListPtr if it converts it to a list object. After appending each element in elemListPtr, Tcl_ListObjAppendList increments the element's reference count since listPtr now also refers to it. For the same reason, Tcl_ListObjAppendElement increments objPtr's reference count. If no error occurs, the two procedures return TCL_OK after appending the objects.
**Tcl_NewListObj** and **Tcl_SetListObj** create a new object or modify an existing object to hold the *objc* elements of the array referenced by *objv* where each element is a pointer to a Tcl object. If *objc* is less than or equal to zero, they return an empty object. The new object's string representation is left invalid. The two procedures increment the reference counts of the elements in *objc* since the list object now refers to them. The new list object returned by **Tcl_NewListObj** has reference count zero.

**Tcl_ListObjGetElements** returns a count and a pointer to an array of the elements in a list object. It returns the count by storing it in the address *objcPtr*. Similarly, it returns the array pointer by storing it in the address *objvPtr*. The memory pointed to is managed by Tcl and should not be freed by the caller. If *listPtr* is not already a list object, **Tcl_ListObjGetElements** will attempt to convert it to one; if the conversion fails, it returns TCL_ERROR and leaves an error message in the interpreter's result object if *interp* is not NULL. Otherwise it returns TCL_OK after storing the count and array pointer.

**Tcl_ListObjLength** returns the number of elements in the list object referenced by *listPtr*. It returns this count by storing an integer in the address *intPtr*. If the object is not already a list object, **Tcl_ListObjLength** will attempt to convert it to one; if the conversion fails, it returns TCL_ERROR and leaves an error message in the interpreter's result object if *interp* is not NULL. Otherwise it returns TCL_OK after storing the list's length.

The procedure **Tcl_ListObjIndex** returns a pointer to the object at element *index* in the list referenced by *listPtr*. It returns this object by storing a pointer to it in the address *objPtrPtr*. If *listPtr* does not already refer to a list object, **Tcl_ListObjIndex** will attempt to convert it to one; if the conversion fails, it returns TCL_ERROR and leaves an error message in the interpreter's result object if *interp* is not NULL. If the index is out of range, that is, *index* is negative or greater than or equal to the number of elements in the list, **Tcl_ListObjIndex** stores a NULL in *objPtrPtr* and returns TCL_OK. Otherwise it returns TCL_OK after storing the element's object pointer. The reference count for the list element is not incremented; the caller must do that if it needs to retain a pointer to the element.

**Tcl_ListObjReplace** replaces zero or more elements of the list referenced by *listPtr* with the *objc* objects in the array referenced by *objv*. If *listPtr* does not point to a list object, **Tcl_ListObjReplace** will attempt to convert it to one; if the conversion fails, it returns TCL_ERROR and leaves an error message in the interpreter's result object if *interp* is not NULL. Otherwise, it returns TCL_OK after replacing the objects. If *objv* is NULL, no new elements are added. If the argument *first* is zero or negative, it refers to the first element. If *first* is greater than or equal to the number of elements in the list, then no elements are deleted; the new elements are appended to the list. *count* gives the number of elements to replace. If *count* is zero or negative then no elements are deleted; the new elements are simply inserted before the one designated by *first*. **Tcl_ListObjReplace** invalidates *listPtr*'s old string representation. The reference counts of any elements inserted from *objv* are incremented since the resulting list now refers to them. Similarly, the reference counts for any replaced objects are decremented.

Because **Tcl_ListObjReplace** combines both element insertion and deletion, it can be used to implement a number of list operations. For example, the following code inserts the *objc* objects referenced by the array of object pointers *objv* just before the element *index* of the list referenced by *listPtr*:

```c
result = Tcl_ListObjReplace(interp, listPtr, index, 0, objc, objv);
```
Similarly, the following code appends the \texttt{objc} objects referenced by the array \texttt{objv} to the end of the list \texttt{listPtr}:

\begin{verbatim}
result = Tcl_ListObjLength(interp, listPtr, \&length);
if (result == TCL_OK) {
    result = Tcl_ListObjReplace(interp, listPtr, length, 0, objc, objv);
}
\end{verbatim}

The \textit{count} list elements starting at \textit{first} can be deleted by simply calling \texttt{Tcl_ListObjReplace} with a NULL \textit{objvPtr}:

\begin{verbatim}
result = Tcl_ListObjReplace(interp, listPtr, first, count, 0, NULL);
\end{verbatim}

\section*{SEE ALSO}

\texttt{Tcl\_NewObj, Tcl\_DecrRefCount, Tcl\_IncrRefCount, Tcl\_GetObjResult}

\section*{KEYWORDS}

append, index, insert, internal representation, length, list, list object, list type, object, object type, replace, string representation

\textit{Copyright © 1996-1997 Sun Microsystems, Inc.}  
\textit{Copyright © 1995-1997 Roger E. Critchlow Jr.}
Macintosh

NAME

Tcl_MacSetEventProc, Tcl_MacConvertTextResource, Tcl_MacEvalResource, Tcl_MacFindResource, Tcl_GetOSTypeFromObj, Tcl_SetOSTypeObj, Tcl_NewOSTypeObj – procedures to handle Macintosh resources and other Macintosh specifics

SYNOPSIS

#include <tcl.h>

int Tcl_MacEvalResource(interp, resourceName, resourceNumber, fileName)
char*
Tcl_MacConvertTextResource(resource)
Handle
Tcl_MacFindResource(interp, resourceType, resourceName, resourceNumber, resFileRef, releaseIt)
Tcl_Obj*
Tcl_NewOSTypeObj(newOSType)
void
Tcl_SetOSTypeObj(objPtr, newOSType)
int
Tcl_GetOSTypeFromObj(interp, objPtr, osTypePtr)
void
Tcl_MacSetEventProc(procPtr)

ARGUMENTS

Tcl_Interp *interp (in)
Interpreter to use for error reporting, or NULL if no error reporting is desired.
CONST char *resourceName (in)
Name of TEXT resource to source, NULL if number should be used.
int resourceNumber (in)
Resource id of source.
CONST char *fileName (in)
Name of file to process. NULL if application resource.
Handle resource (in)
Handle to TEXT resource.
long resourceType (in)
Type of resource to load.
CONST char *resFileRef (in)
Registered resource file reference, NULL if searching all open resource files.
int *releaseIt (out)
Should we release this resource when done.
int newOSType (in)
    Int used to initialize the new object or set the object's value.

Tcl_Obj *objPtr (in)
    Object whose internal representation is to be set or retrieved.

osTypePtr out ()
    Place to store the resulting integer.

Tcl_MacConvertEventProc procPtr (in)
    Reference to the new function to handle all incoming Mac events.

INTRODUCTION

The described routines are used to implement the Macintosh specific resource command and the Mac specific notifier. They manipulate or use Macintosh resources and provide administration for open resource file references.

DESCRIPTION

Tcl_MacEvalResource extends the source command to Macintosh resources. It sources Tcl code from a Text resource. Currently only sources the resource by name, file IDs may be supported at a later date.

Tcl_MacConvertTextResource converts a TEXT resource into a Tcl suitable string. It mallocs the returned memory, converts \r to \n, and appends a null. The caller has the responsibility for freeing the memory.

Tcl_MacFindResource provides a higher level interface for loading resources. It is used by resource read.

Tcl_NewOSTypeObj is used to create a new resource name type object. The object type is "ostype".

Tcl_SetOSTypeObj modifies an object to be a resource type and to have the specified long value.

Tcl_GetOSTypeFromObj attempts to return an int from the Tcl object "objPtr". If the object is not already an int, an attempt will be made to convert it to one.

Tcl_MacSetEventProc sets the event handling procedure for the application. This function will be passed all incoming Mac events. This function usually controls the console or some other entity like Tk.

RESOURCE TYPES

Resource types are 4–byte values used by the macintosh resource facility to tag parts of the resource fork in a file so that the OS knows how to handle them. As all 4 bytes are restricted to printable characters such a type can be interpreted as a 4 character string too.

KEYWORDS

macintosh, mac, resource, notifier
Tcl_Main

NAME

Tcl_Main, Tcl_SetMainLoop – main program and event loop definition for Tcl–based applications

SYNOPSIS

#include <tcl.h>
Tcl_Main(argc, argv, appInitProc)
Tcl_SetMainLoop(mainLoopProc)

ARGUMENTS

int argc (in)
  Number of elements in argv.
char *argv[] (in)
  Array of strings containing command–line arguments.
Tcl_AppInitProc *appInitProc (in)
  Address of an application–specific initialization procedure. The value for this argument is usually Tcl_AppInit.
Tcl_MainLoopProc *mainLoopProc (in)
  Address of an application–specific event loop procedure.

DESCRIPTION

Tcl_Main can serve as the main program for Tcl–based shell applications. A `shell application'' is a program like tclsh or wish that supports both interactive interpretation of Tcl and evaluation of a script contained in a file given as a command line argument. Tcl_Main is offered as a convenience to developers of shell applications, so they do not have to reproduce all of the code for proper initialization of the Tcl library and interactive shell operation. Other styles of embedding Tcl in an application are not supported by Tcl_Main. Those must be achieved by calling lower level functions in the Tcl library directly. The Tcl_Main function has been offered by the Tcl library since release Tcl 7.4. In older releases of Tcl, the Tcl library itself defined a function main, but that lacks flexibility of embedding style and having a function main in a library (particularly a shared library) causes problems on many systems. Having main in the Tcl library would also make it hard to use Tcl in C++ programs, since C++ programs must have special C++ main functions.

Normally each shell application contains a small main function that does nothing but invoke Tcl_Main. Tcl_Main then does all the work of creating and running a tclsh–like application.

Tcl_Main is not provided by the public interface of Tcl's stub library. Programs that call Tcl_Main must be linked against the standard Tcl library. Extensions (stub–enabled or not) are not intended to call Tcl_Main.
Tcl_Main is not thread-safe. It should only be called by a single master thread of a multi-threaded application. This restriction is not a problem with normal use described above.

Tcl_Main and therefore all applications based upon it, like tclsh, use Tcl_GetStdChannel to initialize the standard channels to their default values. See Tcl_StandardChannels for more information.

Tcl_Main supports two modes of operation, depending on the values of argc and argv. If argv[1] exists and does not begin with the character -, it is taken to be the name of a file containing a startup script, which Tcl_Main will attempt to evaluate. Otherwise, Tcl_Main will enter an interactive mode.

In either mode, Tcl_Main will define in its master interpreter the Tcl variables argc, argv, argv0, and tcl_interactive, as described in the documentation for tclsh.

When it has finished its own initialization, but before it processes commands, Tcl_Main calls the procedure given by the appInitProc argument. This procedure provides a "hook" for the application to perform its own initialization of the interpreter created by Tcl_Main, such as defining application-specific commands. The procedure must have an interface that matches the type Tcl_AppInitProc:

```c
typedef int Tcl_AppInitProc(Tcl_Interp *interp);
```

AppInitProc is almost always a pointer to Tcl_AppInit; for more details on this procedure, see the documentation for Tcl_AppInit.

When the appInitProc is finished, Tcl_Main enters one of its two modes. If a startup script has been provided, Tcl_Main attempts to evaluate it. Otherwise, interactive mode begins with examination of the variable tcl_rcFileName in the master interpreter. If that variable exists and holds the name of a readable file, the contents of that file are evaluated in the master interpreter. Then interactive operations begin, with prompts and command evaluation results written to the standard output channel, and commands read from the standard input channel and then evaluated. The prompts written to the standard output channel may be customized by defining the Tcl variables tcl_prompt1 and tcl_prompt2 as described in the documentation for tclsh. The prompts and command evaluation results are written to the standard output channel only if the Tcl variable tcl_interactive in the master interpreter holds a non-zero integer value.

Tcl_SetMainLoop allows setting an event loop procedure to be run. This allows, for example, Tk to be dynamically loaded and set its event loop. The event loop will run following the startup script. If you are in interactive mode, setting the main loop procedure will cause the prompt to become fileevent based and then the loop procedure is called. When the loop procedure returns in interactive mode, interactive operation will continue. The main loop procedure must have an interface that matches the type Tcl_MainLoopProc:

```c
typedef void Tcl_MainLoopProc(void);
```

Tcl_Main does not return. Normally a program based on Tcl_Main will terminate when the exit command is evaluated. In interactive mode, if an EOF or channel error is encountered on the standard input channel, then Tcl_Main itself will evaluate the exit command after the main loop procedure (if any) returns. In non-interactive mode, after Tcl_Main evaluates the startup script, and the main loop procedure (if any) returns, Tcl_Main will also evaluate the exit command.
SEE ALSO

tclsh, Tcl_GetStdChannel, Tcl_StandardChannels, Tcl_AppInit, exit

KEYWORDS

application-specific initialization, command-line arguments, main program

Copyright ' 1994 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 2000 Ajuba Solutions.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
Object

NAME

Tcl_NewObj, Tcl_DuplicateObj, Tcl_IncrRefCount, Tcl_DecrRefCount, Tcl_IsShared, Tcl_InvalidateStringRep – manipulate Tcl objects

SYNOPSIS

#include <tcl.h>
Tcl_Obj *
Tcl_NewObj()
Tcl_Obj *
Tcl_DuplicateObj(objPtr)
Tcl_IncrRefCount(objPtr)
Tcl_DecrRefCount(objPtr)
int
Tcl_IsShared(objPtr)
Tcl_InvalidateStringRep(objPtr)

ARGUMENTS

Tcl_Obj *objPtr (in)
Points to an object; must have been the result of a previous call to Tcl_NewObj.

INTRODUCTION

This man page presents an overview of Tcl objects and how they are used. It also describes generic procedures for managing Tcl objects. These procedures are used to create and copy objects, and increment and decrement the count of references (pointers) to objects. The procedures are used in conjunction with ones that operate on specific types of objects such as Tcl_GetIntFromObj and Tcl_ListObjAppendElement. The individual procedures are described along with the data structures they manipulate.

Tcl's dual–ported objects provide a general–purpose mechanism for storing and exchanging Tcl values. They largely replace the use of strings in Tcl. For example, they are used to store variable values, command arguments, command results, and scripts. Tcl objects behave like strings but also hold an internal representation that can be manipulated more efficiently. For example, a Tcl list is now represented as an object that holds the list's string representation as well as an array of pointers to the objects for each list element. Dual–ported objects avoid most runtime type conversions. They also improve the speed of many operations since an appropriate representation is immediately available. The compiler itself uses Tcl objects to cache the instruction bytecodes resulting from compiling scripts.

The two representations are a cache of each other and are computed lazily. That is, each representation is only computed when necessary, it is computed from the other representation, and, once computed, it is saved. In
addition, a change in one representation invalidates the other one. As an example, a Tcl program doing integer
calculations can operate directly on a variable's internal machine integer representation without having to
constantly convert between integers and strings. Only when it needs a string representing the variable's value,
say to print it, will the program regenerate the string representation from the integer. Although objects contain
an internal representation, their semantics are defined in terms of strings: an up-to-date string can always be
obtained, and any change to the object will be reflected in that string when the object's string representation is
fetched. Because of this representation invalidation and regeneration, it is dangerous for extension writers to
access Tcl_Obj fields directly. It is better to access Tcl_Obj information using procedures like
Tcl_GetStringFromObj and Tcl_GetString.

Objects are allocated on the heap and are referenced using a pointer to their Tcl_Obj structure. Objects are
shared as much as possible. This significantly reduces storage requirements because some objects such as long
lists are very large. Also, most Tcl values are only read and never modified. This is especially true for
procedure arguments, which can be shared between the caller and the called procedure. Assignment and
argument binding is done by simply assigning a pointer to the value. Reference counting is used to determine
when it is safe to reclaim an object's storage.

Tcl objects are typed. An object's internal representation is controlled by its type. Seven types are predefined
in the Tcl core including integer, double, list, and bytecode. Extension writers can extend the set of types by
using the procedure Tcl_RegisterObjType.

THE TCL_OBJ STRUCTURE

Each Tcl object is represented by a Tcl_Obj structure which is defined as follows.

typedef struct Tcl_Obj {
    int refCount;
    char * bytes;
    int length;
    Tcl_Obj * type;
    union {
        long longValue;
        double doubleValue;
        VOID * otherValue;
        struct {
            VOID * ptr1;
            VOID * ptr2;
        } twoPtrValue;
    } internalRep;
} Tcl_Obj;

The bytes and the length members together hold an object's UTF–8 string representation, which is a counted
string not containing null bytes (UTF–8 null characters should be encoded as a two byte sequence: 192, 128.)
bytes points to the first byte of the string representation. The length member gives the number of bytes. The
byte array must always have a null byte after the last data byte, at offset length; this allows string
representations to be treated as conventional null–terminated C strings. C programs use
Tcl_GetStringFromObj and Tcl_GetString to get an object's string representation. If bytes is NULL, the
string representation is invalid.
An object's type manages its internal representation. The member `typePtr` points to the `Tcl_ObjType` structure that describes the type. If `typePtr` is NULL, the internal representation is invalid.

The `internalRep` union member holds an object's internal representation. This is either a (long) integer, a double-precision floating point number, a pointer to a value containing additional information needed by the object's type to represent the object, or two arbitrary pointers.

The `refCount` member is used to tell when it is safe to free an object's storage. It holds the count of active references to the object. Maintaining the correct reference count is a key responsibility of extension writers. Reference counting is discussed below in the section "STORAGE MANAGEMENT OF OBJECTS.

Although extension writers can directly access the members of a Tcl_Obj structure, it is much better to use the appropriate procedures and macros. For example, extension writers should never read or update `refCount` directly; they should use macros such as `Tcl_IncrRefCount` and `Tcl_IsShared` instead.

A key property of Tcl objects is that they hold two representations. An object typically starts out containing only a string representation: it is untyped and has a NULL `typePtr`. An object containing an empty string or a copy of a specified string is created using `Tcl_NewObj` or `Tcl_NewStringObj` respectively. An object's string value is gotten with `Tcl_GetStringFromObj` or `Tcl_GetString` and changed with `Tcl_SetStringObj`. If the object is later passed to a procedure like `Tcl_GetIntFromObj` that requires a specific internal representation, the procedure will create one and set the object's `typePtr`. The internal representation is computed from the string representation. An object's two representations are duals of each other: changes made to one are reflected in the other. For example, `Tcl_ListObjReplace` will modify an object's internal representation and the next call to `Tcl_GetStringFromObj` or `Tcl_GetString` will reflect that change.

Representations are recomputed lazily for efficiency. A change to one representation made by a procedure such as `Tcl_ListObjReplace` is not reflected immediately in the other representation. Instead, the other representation is marked invalid so that it is only regenerated if it is needed later. Most C programmers never have to be concerned with how this is done and simply use procedures such as `Tcl_GetBooleanFromObj` or `Tcl_ListObjIndex`. Programmers that implement their own object types must check for invalid representations and mark representations invalid when necessary. The procedure `Tcl_InvalidateStringRep` is used to mark an object's string representation invalid and to free any storage associated with the old string representation.

Objects usually remain one type over their life, but occasionally an object must be converted from one type to another. For example, a C program might build up a string in an object with repeated calls to `Tcl_AppendToObj`, and then call `Tcl_ListObjIndex` to extract a list element from the object. The same object holding the same string value can have several different internal representations at different times. Extension writers can also force an object to be converted from one type to another using the `Tcl_ConvertToType` procedure. Only programmers that create new object types need to be concerned about how this is done. A procedure defined as part of the object type's implementation creates a new internal representation for an object and changes its `typePtr`. See the man page for `Tcl_RegisterObjType` to see how to create a new object type.
EXAMPLE OF THE LIFETIME OF AN OBJECT

As an example of the lifetime of an object, consider the following sequence of commands:

set x 123

This assigns to \textit{x} an untyped object whose \textit{bytes} member points to 123 and \textit{length} member contains 3. The object's \textit{typePtr} member is NULL.

\texttt{puts "x is $x"}

\textit{x}'s string representation is valid (since \textit{bytes} is non-NULL) and is fetched for the command.

\texttt{incr x}

The \texttt{incr} command first gets an integer from \textit{x}'s object by calling \texttt{Tcl\_GetIntFromObj}. This procedure checks whether the object is already an integer object. Since it is not, it converts the object by setting the object's \textit{internalRep.longValue} member to the integer 123 and setting the object's \textit{typePtr} to point to the integer \texttt{Tcl\_ObjType} structure. Both representations are now valid. \texttt{incr} increments the object's integer internal representation then invalidates its string representation (by calling \texttt{Tcl\_InvalidateStringRep}) since the string representation no longer corresponds to the internal representation.

\texttt{puts "x is now $x"}

The string representation of \textit{x}'s object is needed and is recomputed. The string representation is now 124. and both representations are again valid.

STORAGE MANAGEMENT OF OBJECTS

Tcl objects are allocated on the heap and are shared as much as possible to reduce storage requirements. Reference counting is used to determine when an object is no longer needed and can safely be freed. An object just created by \texttt{Tcl\_NewObj} or \texttt{Tcl\_NewStringObj} has \textit{refCount} 0. The macro \texttt{Tcl\_IncrRefCount} increments the reference count when a new reference to the object is created. The macro \texttt{Tcl\_DecrRefCount} decrements the count when a reference is no longer needed and, if the object's reference count drops to zero, frees its storage. An object shared by different code or data structures has \textit{refCount} greater than 1. Incrementing an object's reference count ensures that it won't be freed too early or have its value change accidentally.

As an example, the bytecode interpreter shares argument objects between calling and called Tcl procedures to avoid having to copy objects. It assigns the call's argument objects to the procedure's formal parameter variables. In doing so, it calls \texttt{Tcl\_IncrRefCount} to increment the reference count of each argument since there is now a new reference to it from the formal parameter. When the called procedure returns, the interpreter calls \texttt{Tcl\_DecrRefCount} to decrement each argument's reference count. When an object's reference count drops less than or equal to zero, \texttt{Tcl\_DecrRefCount} reclaims its storage. Most command procedures do not have to be concerned about reference counting since they use an object's value immediately and don't retain a pointer to the object after they return. However, if they do retain a pointer to an object in a
data structure, they must be careful to increment its reference count since the retained pointer is a new reference.

Command procedures that directly modify objects such as those for `lappend` and `linsert` must be careful to copy a shared object before changing it. They must first check whether the object is shared by calling `Tcl_IsShared`. If the object is shared they must copy the object by using `Tcl_DuplicateObj`; this returns a new duplicate of the original object that has refCount 0. If the object is not shared, the command procedure "owns" the object and can safely modify it directly. For example, the following code appears in the command procedure that implements `linsert`. This procedure modifies the list object passed to it in `objv[1]` by inserting `objc−3` new elements before `index`.

```c
listPtr = objv[1];
if (Tcl_IsShared(listPtr)) {
    listPtr = Tcl_DuplicateObj(listPtr);
}
result = Tcl_ListObjReplace(interp, listPtr, index, 0, (objc−3), &(objv[3]));
```

As another example, `incr`'s command procedure must check whether the variable's object is shared before incrementing the integer in its internal representation. If it is shared, it needs to duplicate the object in order to avoid accidently changing values in other data structures.

**SEE ALSO**

`Tcl_ConvertToType`, `Tcl_GetIntFromObj`, `Tcl_ListObjAppendElement`, `Tcl_ListObjIndex`, `Tcl_ListObjReplace`, `Tcl_RegisterObjType`

**KEYWORDS**

internal representation, object, object creation, object type, reference counting, string representation, type conversion

---

Copyright ' 1996-1997 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
ObjectType

NAME

Tcl_RegisterObjType, Tcl_GetObjType, Tcl_AppendAllObjTypes, Tcl_ConvertToType – manipulate Tcl object types

SYNOPSIS

#include <tcl.h>

Tcl_RegisterObjType(typePtr)
Tcl_ObjType *
Tcl_GetObjType(typeName)
int
Tcl_AppendAllObjTypes(interp, objPtr)
int
Tcl_ConvertToType(interp, objPtr, typePtr)

ARGUMENTS

Tcl_ObjType *typePtr (in)
Points to the structure containing information about the Tcl object type. This storage must live forever, typically by being statically allocated.

CONST char *typeName (in)
The name of a Tcl object type that Tcl_GetObjType should look up.

Tcl_Interp *interp (in)
Interpreter to use for error reporting.

Tcl_Obj *objPtr (in)
For Tcl_AppendAllObjTypes, this points to the object onto which it appends the name of each object type as a list element. For Tcl_ConvertToType, this points to an object that must have been the result of a previous call to Tcl_NewObj.

DESCRIPTION

The procedures in this man page manage Tcl object types. They are used to register new object types, look up types, and force conversions from one type to another.

Tcl_RegisterObjType registers a new Tcl object type in the table of all object types supported by Tcl. The argument typePtr points to a Tcl_ObjType structure that describes the new type by giving its name and by supplying pointers to four procedures that implement the type. If the type table already contains a type with the same name as in typePtr, it is replaced with the new type. The Tcl_ObjType structure is described in the section THE TCL_OBJTYPE STRUCTURE below.
Tcl_GetObjType returns a pointer to the Tcl_ObjType with name typeName. It returns NULL if no type with that name is registered.

Tcl_AppendAllObjTypes appends the name of each object type as a list element onto the Tcl object referenced by objPtr. The return value is TCL_OK unless there was an error converting objPtr to a list object; in that case TCL_ERROR is returned.

Tcl_ConvertToType converts an object from one type to another if possible. It creates a new internal representation for objPtr appropriate for the target type typePtr and sets its typePtr member to that type. Any internal representation for objPtr's old type is freed. If an error occurs during conversion, it returns TCL_ERROR and leaves an error message in the result object for interp unless interp is NULL. Otherwise, it returns TCL_OK. Passing a NULL interp allows this procedure to be used as a test whether the conversion can be done (and in fact was done).

THE TCL_OBJTYPE STRUCTURE

Extension writers can define new object types by defining four procedures, initializing a Tcl_ObjType structure to describe the type, and calling Tcl_RegisterObjType. The Tcl_ObjType structure is defined as follows:

```c
typedef struct Tcl_ObjType {
    char *name;
    Tcl_FreeInternalRepProc *freeIntRepProc;
    Tcl_DupInternalRepProc *dupIntRepProc;
    Tcl_UpdateStringProc *updateStringProc;
    Tcl_SetFromAnyProc *setFromAnyProc;
} Tcl_ObjType;
```

The name member describes the name of the type, e.g. int. Extension writers can look up an object type using its name with the Tcl_GetObjType procedure. The remaining four members are pointers to procedures called by the generic Tcl object code:

The setFromAnyProc member contains the address of a function called to create a valid internal representation from an object’s string representation.

```c
typedef int (Tcl_SetFromAnyProc) (Tcl_Interp *interp, Tcl_Obj *objPtr);
```

If an internal representation can't be created from the string, it returns TCL_ERROR and puts a message describing the error in the result object for interp unless interp is NULL. If setFromAnyProc is successful, it stores the new internal representation, sets objPtr's typePtr member to point to setFromAnyProc's Tcl_ObjType, and returns TCL_OK. Before setting the new internal representation, the setFromAnyProc must free any internal representation of objPtr's old type; it does this by calling the old type's freeIntRepProc if it is not NULL. As an example, the setFromAnyProc for the builtin Tcl integer type gets an up-to-date string representation for objPtr by calling Tcl_GetStringFromObj. It parses the string to obtain an integer and, if this succeeds, stores the integer in objPtr's internal representation and sets objPtr's typePtr member to point to the integer type's Tcl_ObjType structure. Do not release objPtr's old internal representation unless you replace it with a new one or reset the typePtr member to NULL.
The `updateStringProc` member contains the address of a function called to create a valid string representation from an object's internal representation.

```c
typedef void (Tcl_UpdateStringProc) (Tcl_Obj *objPtr);
```

`objPtr`'s `bytes` member is always NULL when it is called. It must always set `bytes` non-NULL before returning. We require the string representation's byte array to have a null after the last byte, at offset `length`; this allows string representations that do not contain null bytes to be treated as conventional null character-terminated C strings. Storage for the byte array must be allocated in the heap by `Tcl_Alloc` or `ckalloc`. Note that `updateStringProc`s must allocate enough storage for the string's bytes and the terminating null byte. The `updateStringProc` for Tcl's builtin list type, for example, builds an array of strings for each element object and then calls `Tcl_Merge` to construct a string with proper Tcl list structure. It stores this string as the list object's string representation.

The `dupIntRepProc` member contains the address of a function called to copy an internal representation from one object to another.

```c
typedef void (Tcl_DupInternalRepProc) (Tcl_Obj *srcPtr, Tcl_Obj *dupPtr);
```

dupPtr's internal representation is made a copy of srcPtr's internal representation. Before the call, srcPtr's internal representation is valid and dupPtr's is not. srcPtr's object type determines what copying its internal representation means. For example, the `dupIntRepProc` for the Tcl integer type simply copies an integer. The builtin list type's `dupIntRepProc` allocates a new array that points at the original element objects; the elements are shared between the two lists (and their reference counts are incremented to reflect the new references).

The `freeIntRepProc` member contains the address of a function that is called when an object is freed.

```c
typedef void (Tcl_FreeInternalRepProc) (Tcl_Obj *objPtr);
```

The `freeIntRepProc` function can deallocate the storage for the object's internal representation and do other type-specific processing necessary when an object is freed. For example, Tcl list objects have an `internalRep.otherValuePtr` that points to an array of pointers to each element in the list. The list type's `freeIntRepProc` decrements the reference count for each element object (since the list will no longer refer to those objects), then deallocates the storage for the array of pointers. The `freeIntRepProc` member can be set to NULL to indicate that the internal representation does not require freeing.

**SEE ALSO**

[Tcl_NewObj](#), [Tcl_DecrRefCount](#), [Tcl_IncrRefCount](#)

**KEYWORDS**

internal representation, object, object type, string representation, type conversion

---

Copyright ' 1996-1997 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
OpenFileChnl

NAME

Tcl_OpenFileChannel, Tcl_OpenCommandChannel, Tcl_MakeFileChannel,
Tcl_GetChannel, Tcl_GetChannelNames, Tcl_GetChannelNamesEx, Tcl_RegisterChannel,
Tcl_UnregisterChannel, Tcl_DetachChannel, Tcl_IsStandardChannel, Tcl_Close,
Tcl_ReadChars, Tcl_Read, Tcl_GetObj, Tcl_Get, Tcl_WriteObj, Tcl_WriteChars,
Tcl_Write, Tcl_Flush, Tcl_Seek, Tcl_Tell, Tcl_GetChannelOption, Tcl_SetChannelOption,
Tcl_Eof, Tcl_InputBlocked, Tcl_InputBuffered, Tcl_OutputBuffered, Tcl_Ungets,
Tcl_ReadRaw, Tcl_WriteRaw – buffered I/O facilities using channels

SYNOPSIS

ARGUMENTS

DESCRIPTION

PLATFORM ISSUES

SEE ALSO

KEYWORDS
NAME

Tcl_OpenFileChannel, Tcl_OpenCommandChannel, Tcl_MakeFileChannel, Tcl_GetChannel,
Tcl_GetChannelNames, Tcl_GetChannelNamesEx, Tcl_RegisterChannel, Tcl_UnregisterChannel,
Tcl_DetachChannel, Tcl_IsStandardChannel, Tcl_Close, Tcl_ReadChars, Tcl_Read, Tcl_GetsObj, Tcl_Gets,
Tcl_WriteObj, Tcl_WriteChars, Tcl_Write, Tcl_Flush, Tcl_Seek, Tcl_Tell, Tcl_GetChannelOption,
Tcl_SetChannelOption, Tcl_Eof, Tcl_InputBlocked, Tcl_InputBuffered, Tcl_OutputBuffered, Tcl_Ungets,
Tcl_ReadRaw, Tcl_WriteRaw – buffered I/O facilities using channels

SYNOPSIS

#include <tcl.h>
Tcl_Channel
Tcl_OpenFileChannel(interp, fileName, mode, permissions)
Tcl_Channel
Tcl_OpenCommandChannel(interp, argc, argv, flags)
Tcl_Channel
Tcl_MakeFileChannel(handle, readOrWrite)
Tcl_Channel
Tcl_GetChannel(interp, channelName, modePtr)
t
Tcl_GetChannelNames(interp)
t
Tcl_GetChannelNamesEx(interp, pattern)
void
Tcl_RegisterChannel(interp, channel)
t
Tcl_UnregisterChannel(interp, channel)
t
Tcl_DetachChannel(interp, channel)
t
Tcl_IsStandardChannel(channel)
t
Tcl_Close(interp, channel)
t
Tcl_ReadChars(channel, readObjPtr, charsToRead, appendFlag)
t
Tcl_Read(channel, readBuf, bytesToRead)
t
Tcl_GetsObj(channel, lineObjPtr)
t
Tcl_Gets(channel, lineRead)
t
Tcl_Ungets(channel, input, inputLen, addAtEnd)
t
Tcl_WriteObj(channel, writeObjPtr)
int
Tcl_WriteChars(channel, charBuf, bytesToWrite)
int
Tcl_Write(channel, byteBuf, bytesToWrite)
int
Tcl_ReadRaw(channel, readBuf, bytesToRead)
int
Tcl_WriteRaw(channel, byteBuf, bytesToWrite)
int
Tcl_Eof(channel)
int
Tcl_Flush(channel)
int
Tcl_InputBlocked(channel)
int
Tcl_InputBuffered(channel)
int
Tcl_OutputBuffered(channel)
Tcl_WideInt
Tcl_Selek(channel, offset, seekMode)
Tcl_WideInt
Tcl_Tell(channel)
int
Tcl_GetChannelOption(interp, channel, optionName, optionValue)
int
Tcl_SetChannelOption(interp, channel, optionName, newValue)

ARGUMENTS

Tcl_Interp *interp (in)
Used for error reporting and to look up a channel registered in it.
CONST char *fileName (in)
The name of a local or network file.
CONST char *mode (in)
Specifies how the file is to be accessed. May have any of the values allowed for the mode argument to
the Tcl open command.
int permissions (in)
POSIX−style permission flags such as 0644. If a new file is created, these permissions will be set on
the created file.
int argc (in)
The number of elements in argv.
CONST char **argv (in)
Arguments for constructing a command pipeline. These values have the same meaning as the
non−switch arguments to the Tcl exec command.
int **flags** *(in)*  
Specifies the disposition of the stdio handles in pipeline: OR−ed combination of **TCL_STDIN**, **TCL_STDOUT**, **TCL_STDERR**, and **TCL_ENFORCE_MODE**. If **TCL_STDIN** is set, stdin for the first child in the pipe is the pipe channel, otherwise it is the same as the standard input of the invoking process; likewise for **TCL_STDOUT** and **TCL_STDERR**. If **TCL_ENFORCE_MODE** is not set, then the pipe can redirect stdio handles to override the stdio handles for which **TCL_STDIN**, **TCL_STDOUT** and **TCL_STDERR** have been set. If it is set, then such redirections cause an error.

ClientData *handle* *(in)*  
Operating system specific handle for I/O to a file. For Unix this is a file descriptor, for Windows it is a HANDLE.

int **readOrWrite** *(in)*  
OR−ed combination of **TCL_READABLE** and **TCL_WRITABLE** to indicate what operations are valid on *handle*.

**CONST char** *channelName* *(in)*  
The name of the channel.

int **modePtr** *(out)*  
Points at an integer variable that will receive an OR−ed combination of **TCL_READABLE** and **TCL_WRITABLE** denoting whether the channel is open for reading and writing.

**CONST char** *pattern* *(in)*  
The pattern to match on, passed to **Tcl_StringMatch**, or NULL.

**Tcl_Channel** *channel* *(in)*  
A Tcl channel for input or output. Must have been the return value from a procedure such as **Tcl_OpenFileChannel**.

**Tcl_Obj** *readObjPtr* *(in/out)*  
A pointer to a Tcl Object in which to store the characters read from the channel.

int **charsToRead** *(in)*  
The number of characters to read from the channel. If the channel's encoding is binary, this is equivalent to the number of bytes to read from the channel.

int **appendFlag** *(in)*  
If non−zero, data read from the channel will be appended to the object. Otherwise, the data will replace the existing contents of the object.

char *readBuf* *(out)*  
A buffer in which to store the bytes read from the channel.

int **bytesToRead** *(in)*  
The number of bytes to read from the channel. The buffer *readBuf* must be large enough to hold this many bytes.

**Tcl_Obj** *lineObjPtr* *(in/out)*  
A pointer to a Tcl object in which to store the line read from the channel. The line read will be appended to the current value of the object.

**Tcl_DString** *lineRead* *(in/out)*  
A pointer to a Tcl dynamic string in which to store the line read from the channel. Must have been initialized by the caller. The line read will be appended to any data already in the dynamic string.

**CONST char** *input* *(in)*  
The input to add to a channel buffer.

int **inputLen** *(in)*
Length of the input

**`int addAtEnd (in)`**
- Flag indicating whether the input should be added to the end or beginning of the channel buffer.

**`Tcl_Obj *writeObjPtr (in)`**
- A pointer to a Tcl Object whose contents will be output to the channel.

**`CONST char *charBuf (in)`**
- A buffer containing the characters to output to the channel.

**`CONST char *byteBuf (in)`**
- A buffer containing the bytes to output to the channel.

**`int bytesToWrite (in)`**
- The number of bytes to consume from `charBuf` or `byteBuf` and output to the channel.

**`Tcl_WideInt offset (in)`**
- How far to move the access point in the channel at which the next input or output operation will be applied, measured in bytes from the position given by `seekMode`. May be either positive or negative.

**`int seekMode (in)`**
- Relative to which point to seek; used with `offset` to calculate the new access point for the channel.
- Legal values are `SEEK_SET`, `SEEK_CUR`, and `SEEK_END`.

**`CONST char *optionName (in)`**
- The name of an option applicable to this channel, such as `−blocking`. May have any of the values accepted by the `fconfigure` command.

**`Tcl_DString *optionValue (in)`**
- Where to store the value of an option or a list of all options and their values. Must have been initialized by the caller.

**`CONST char *newValue (in)`**
- New value for the option given by `optionName`.

### DESCRIPTION

The Tcl channel mechanism provides a device-independent and platform-independent mechanism for performing buffered input and output operations on a variety of file, socket, and device types. The channel mechanism is extensible to new channel types, by providing a low level channel driver for the new type; the channel driver interface is described in the manual entry for `Tcl_CreateChannel`. The channel mechanism provides a buffering scheme modeled after Unix's standard I/O, and it also allows for nonblocking I/O on channels.

The procedures described in this manual entry comprise the C APIs of the generic layer of the channel architecture. For a description of the channel driver architecture and how to implement channel drivers for new types of channels, see the manual entry for `Tcl_CreateChannel`.

### TCL_OPENFILECHANNEL

**`Tcl_OpenFileChannel`** opens a file specified by `fileName` and returns a channel handle that can be used to perform input and output on the file. This API is modeled after the `fopen` procedure of the Unix standard I/O library. The syntax and meaning of all arguments is similar to those given in the Tcl `open` command when opening a file. If an error occurs while opening the channel, `Tcl_OpenFileChannel` returns NULL and
records a POSIX error code that can be retrieved with \texttt{Tcl\_GetErrno}. In addition, if \texttt{interp} is non–NULL, \texttt{Tcl\_OpenFileChannel} leaves an error message in \texttt{interp}'s result after any error. As of Tcl 8.4, the object–based API \texttt{Tcl\_FSOpenFileChannel} should be used in preference to \texttt{Tcl\_OpenFileChannel} wherever possible.

The newly created channel is not registered in the supplied interpreter; to register it, use \texttt{Tcl\_RegisterChannel}, described below. If one of the standard channels, \texttt{stdin}, \texttt{stdout} or \texttt{stderr} was previously closed, the act of creating the new channel also assigns it as a replacement for the standard channel.

\section*{TCL\_OPENCOMMANDCHANNEL}

\texttt{Tcl\_OpenCommandChannel} provides a C–level interface to the functions of the \texttt{exec} and \texttt{open} commands. It creates a sequence of subprocesses specified by the \texttt{argv} and \texttt{argc} arguments and returns a channel that can be used to communicate with these subprocesses. The \texttt{flags} argument indicates what sort of communication will exist with the command pipeline.

If the \texttt{TCL\_STDIN} flag is set then the standard input for the first subprocess will be tied to the channel: writing to the channel will provide input to the subprocess. If \texttt{TCL\_STDIN} is not set, then standard input for the first subprocess will be the same as this application's standard input. If \texttt{TCL\_STDOUT} is set then standard output from the last subprocess can be read from the channel; otherwise it goes to this application's standard output. If \texttt{TCL\_STDERR} is set, standard error output for all subprocesses is returned to the channel and results in an error when the channel is closed; otherwise it goes to this application's standard error. If \texttt{TCL\_ENFORCE\_MODE} is not set, then \texttt{argc} and \texttt{argv} can redirect the stdio handles to override \texttt{TCL\_STDIN}, \texttt{TCL\_STDOUT}, and \texttt{TCL\_STDERR}; if it is set, then it is an error for \texttt{argc} and \texttt{argv} to override stdio channels for which \texttt{TCL\_STDIN}, \texttt{TCL\_STDOUT}, and \texttt{TCL\_STDERR} have been set.

If an error occurs while opening the channel, \texttt{Tcl\_OpenCommandChannel} returns NULL and records a POSIX error code that can be retrieved with \texttt{Tcl\_GetErrno}. In addition, \texttt{Tcl\_OpenCommandChannel} leaves an error message in the interpreter's result if \texttt{interp} is not NULL.

The newly created channel is not registered in the supplied interpreter; to register it, use \texttt{Tcl\_RegisterChannel}, described below. If one of the standard channels, \texttt{stdin}, \texttt{stdout} or \texttt{stderr} was previously closed, the act of creating the new channel also assigns it as a replacement for the standard channel.

\section*{TCL\_MAKEFILECHANNEL}

\texttt{Tcl\_MakeFileChannel} makes a \texttt{Tcl\_Channel} from an existing, platform–specific, file handle. The newly created channel is not registered in the supplied interpreter; to register it, use \texttt{Tcl\_RegisterChannel}, described below. If one of the standard channels, \texttt{stdin}, \texttt{stdout} or \texttt{stderr} was previously closed, the act of creating the new channel also assigns it as a replacement for the standard channel.
**TCL_GETCHANNEL**

*Tcl_GetChannel* returns a channel given the *channelName* used to create it with *Tcl_CreateChannel* and a pointer to a Tcl interpreter in *interp*. If a channel by that name is not registered in that interpreter, the procedure returns NULL. If the *modePtr* argument is not NULL, it points at an integer variable that will receive an OR-ed combination of *TCL_READABLE* and *TCL_WRITABLE* describing whether the channel is open for reading and writing.

*Tcl_GetChannelNames* and *Tcl_GetChannelNamesEx* write the names of the registered channels to the interpreter's result as a list object. *Tcl_GetChannelNamesEx* will filter these names according to the *pattern*. If *pattern* is NULL, then it will not do any filtering. The return value is *TCL_OK* if no errors occurred writing to the result, otherwise it is *TCL_ERROR*, and the error message is left in the interpreter's result.

**TCL_REGISTERCHANNEL**

*Tcl_RegisterChannel* adds a channel to the set of channels accessible in *interp*. After this call, Tcl programs executing in that interpreter can refer to the channel in input or output operations using the name given in the call to *Tcl_CreateChannel*. After this call, the channel becomes the property of the interpreter, and the caller should not call *Tcl_Close* for the channel; the channel will be closed automatically when it is unregistered from the interpreter.

Code executing outside of any Tcl interpreter can call *Tcl_RegisterChannel* with *interp* as NULL, to indicate that it wishes to hold a reference to this channel. Subsequently, the channel can be registered in a Tcl interpreter and it will only be closed when the matching number of calls to *Tcl_UnregisterChannel* have been made. This allows code executing outside of any interpreter to safely hold a reference to a channel that is also registered in a Tcl interpreter.

This procedure interacts with the code managing the standard channels. If no standard channels were initialized before the first call to *Tcl_RegisterChannel* they will get initialized by that call. See *Tcl_StandardChannels* for a general treatise about standard channels and the behaviour of the Tcl library with regard to them.

**TCL_UNREGISTERCHANNEL**

*Tcl_UnregisterChannel* removes a channel from the set of channels accessible in *interp*. After this call, Tcl programs will no longer be able to use the channel's name to refer to the channel in that interpreter. If this operation removed the last registration of the channel in any interpreter, the channel is also closed and destroyed.

Code not associated with a Tcl interpreter can call *Tcl_UnregisterChannel* with *interp* as NULL, to indicate to Tcl that it no longer holds a reference to that channel. If this is the last reference to the channel, it will now be closed. *Tcl_UnregisterChannel* is very similar to *Tcl_DetachChannel* except that it will also close the channel if no further references to it exist.
**TCL_DETACHCHANNEL**

*Tcl_DetachChannel* removes a channel from the set of channels accessible in *interp*. After this call, Tcl programs will no longer be able to use the channel's name to refer to the channel in that interpreter. Beyond that, this command has no further effect. It cannot be used on the standard channels (stdout, stderr, stdin), and will return TCL_ERROR if passed one of those channels.

Code not associated with a Tcl interpreter can call *Tcl_DetachChannel* with *interp* as NULL, to indicate to Tcl that it no longer holds a reference to that channel. If this is the last reference to the channel, unlike *Tcl_UnregisterChannel*, it will not be closed.

**TCL_ISSTANDARDCHANNEL**

*Tcl_IsStandardChannel* tests whether a channel is one of the three standard channels, stdin, stdout or stderr. If so, it returns 1, otherwise 0.

No attempt is made to check whether the given channel or the standard channels are initialized or otherwise valid.

**TCL_CLOSE**

*Tcl_Close* destroys the channel *channel*, which must denote a currently open channel. The channel should not be registered in any interpreter when *Tcl_Close* is called. Buffered output is flushed to the channel's output device prior to destroying the channel, and any buffered input is discarded. If this is a blocking channel, the call does not return until all buffered data is successfully sent to the channel's output device. If this is a nonblocking channel and there is buffered output that cannot be written without blocking, the call returns immediately; output is flushed in the background and the channel will be closed once all of the buffered data has been output. In this case errors during flushing are not reported.

If the channel was closed successfully, *Tcl_Close* returns TCL_OK. If an error occurs, *Tcl_Close* returns TCL_ERROR and records a POSIX error code that can be retrieved with *Tcl_GetErrno*. If the channel is being closed synchronously and an error occurs during closing of the channel and *interp* is not NULL, an error message is left in the interpreter's result.

Note: it is not safe to call *Tcl_Close* on a channel that has been registered using *Tcl_RegisterChannel*; see the documentation for *Tcl_RegisterChannel*, above, for details. If the channel has ever been given as the *chan* argument in a call to *Tcl_RegisterChannel*, you should instead use *Tcl_UnregisterChannel*, which will internally call *Tcl_Close* when all calls to *Tcl_RegisterChannel* have been matched by corresponding calls to *Tcl_UnregisterChannel*.

**TCL_READCHARS AND TCL_READ**

*Tcl_ReadChars* consumes bytes from *channel*, converting the bytes to UTF-8 based on the channel's encoding and storing the produced data in *readObjPtr*'s string representation. The return value of *Tcl_ReadChars* is the number of characters, up to *charsToRead*, that were stored in *readObjPtr*. If an error
occurs while reading, the return value is −1 and **Tcl_ReadChars** records a POSIX error code that can be retrieved with **Tcl_GetErrno**.

Setting **charsToRead** to −1 will cause the command to read all characters currently available (non–blocking) or everything until eof (blocking mode).

The return value may be smaller than the value to read, indicating that less data than requested was available. This is called a short read. In blocking mode, this can only happen on an end–of–file. In nonblocking mode, a short read can also occur if there is not enough input currently available: **Tcl_ReadChars** returns a short count rather than waiting for more data.

If the channel is in blocking mode, a return value of zero indicates an end–of–file condition. If the channel is in nonblocking mode, a return value of zero indicates either that no input is currently available or an end–of–file condition. Use **Tcl_Eof** and **Tcl_InputBlocked** to tell which of these conditions actually occurred.

**Tcl_ReadChars** translates the various end–of–line representations into the canonical \n internal representation according to the current end–of–line recognition mode. End–of–line recognition and the various platform–specific modes are described in the manual entry for the Tcl **fconfigure** command.

As a performance optimization, when reading from a channel with the encoding **binary**, the bytes are not converted to UTF–8 as they are read. Instead, they are stored in **readObjPtr**'s internal representation as a byte–array object. The string representation of this object will only be constructed if it is needed (e.g., because of a call to **Tcl_GetStringFromObj**). In this way, byte–oriented data can be read from a channel, manipulated by calling **Tcl_GetByteArrayFromObj** and related functions, and then written to a channel without the expense of ever converting to or from UTF–8.

**Tcl_Read** is similar to **Tcl_ReadChars**, except that it doesn't do encoding conversions, regardless of the channel's encoding. It is deprecated and exists for backwards compatibility with non–internationalized Tcl extensions. It consumes bytes from **channel** and stores them in **readBuf**, performing end–of–line translations on the way. The return value of **Tcl_Read** is the number of bytes, up to **bytesToRead**, written in **readBuf**. The buffer produced by **Tcl_Read** is not null–terminated. Its contents are valid from the zeroth position up to and excluding the position indicated by the return value.

**Tcl_ReadRaw** is the same as **Tcl_Read** but does not compensate for stacking. While **Tcl_Read** (and the other functions in the API) always get their data from the topmost channel in the stack the supplied channel is part of, **Tcl_ReadRaw** does not. Thus this function is only usable for transformational channel drivers, i.e. drivers used in the middle of a stack of channels, to move data from the channel below into the transformation.

**TCL_GETSOBJ AND TCL_GETS**

**Tcl_GetObj** consumes bytes from **channel**, converting the bytes to UTF–8 based on the channel's encoding, until a full line of input has been seen. If the channel's encoding is **binary**, each byte read from the channel is treated as an individual Unicode character. All of the characters of the line except for the terminating
end–of–line character(s) are appended to lineObjPtr's string representation. The end–of–line character(s) are read and discarded.

If a line was successfully read, the return value is greater than or equal to zero and indicates the number of bytes stored in lineObjPtr. If an error occurs, Tcl_GetsObj returns −1 and records a POSIX error code that can be retrieved with Tcl_GetErrno. Tcl_GetsObj also returns −1 if the end of the file is reached; the Tcl_EOF procedure can be used to distinguish an error from an end–of–file condition.

If the channel is in nonblocking mode, the return value can also be −1 if no data was available or the data that was available did not contain an end–of–line character. When −1 is returned, the Tcl_InputBlocked procedure may be invoked to determine if the channel is blocked because of input unavailability.

Tcl_Gets is the same as Tcl_GetsObj except the resulting characters are appended to the dynamic string given by lineRead rather than a Tcl object.

TCL_UNGETS

Tcl_Ungets is used to add data to the input queue of a channel, at either the head or tail of the queue. The pointer input points to the data that is to be added. The length of the input to add is given by inputLen. A non–zero value of addAtEnd indicates that the data is to be added at the end of queue; otherwise it will be added at the head of the queue. If channel has a "sticky" EOF set, no data will be added to the input queue. Tcl_Ungets returns inputLen or −1 if an error occurs.

TCL_WRITECHARS, TCL_WRITEOBJ, AND TCL_WRITE

Tcl_WriteChars accepts bytesToWrite bytes of character data at charBuf. The UTF–8 characters in the buffer are converted to the channel's encoding and queued for output to channel. If bytesToWrite is negative, Tcl_WriteChars expects charBuf to be null–terminated and it outputs everything up to the null.

Data queued for output may not appear on the output device immediately, due to internal buffering. If the data should appear immediately, call Tcl_Flush after the call to Tcl_WriteChars, or set the −buffering option on the channel to none. If you wish the data to appear as soon as a complete line is accepted for output, set the −buffering option on the channel to line mode.

The return value of Tcl_WriteChars is a count of how many bytes were accepted for output to the channel. This is either greater than zero to indicate success or −1 to indicate that an error occurred. If an error occurs, Tcl_WriteChars records a POSIX error code that may be retrieved with Tcl_GetErrno.

Newline characters in the output data are translated to platform–specific end–of–line sequences according to the −translation option for the channel. This is done even if the channel has no encoding.

Tcl_WriteObj is similar to Tcl_WriteChars except it accepts a Tcl object whose contents will be output to the channel. The UTF–8 characters in writeObjPtr's string representation are converted to the channel's encoding and queued for output to channel. As a performance optimization, when writing to a channel with the encoding binary, UTF–8 characters are not converted as they are written. Instead, the bytes in
writeObjPtr's internal representation as a byte-array object are written to the channel. The byte-array representation of the object will be constructed if it is needed. In this way, byte-oriented data can be read from a channel, manipulated by calling `Tcl_GetByteArrayFromObj` and related functions, and then written to a channel without the expense of ever converting to or from UTF-8.

`Tcl_Write` is similar to `Tcl_WriteChars` except that it doesn't do encoding conversions, regardless of the channel's encoding. It is deprecated and exists for backwards compatibility with non-internationalized Tcl extensions. It accepts `bytesToWrite` bytes of data at `byteBuf` and queues them for output to `channel`. If `bytesToWrite` is negative, `Tcl_Write` expects `byteBuf` to be null-terminated and it outputs everything up to the null.

`Tcl_WriteRaw` is the same as `Tcl_Write` but does not compensate for stacking. While `Tcl_Write` (and the other functions in the API) always feed their input to the topmost channel in the stack the supplied channel is part of, `Tcl_WriteRaw` does not. Thus this function is only usable for transformational channel drivers, i.e. drivers used in the middle of a stack of channels, to move data from the transformation into the channel below it.

**TCL_FLUSH**

`Tcl_Flush` causes all of the buffered output data for `channel` to be written to its underlying file or device as soon as possible. If the channel is in blocking mode, the call does not return until all the buffered data has been sent to the channel or some error occurred. The call returns immediately if the channel is nonblocking; it starts a background flush that will write the buffered data to the channel eventually, as fast as the channel is able to absorb it.

The return value is normally `TCL_OK`. If an error occurs, `Tcl_Flush` returns `TCL_ERROR` and records a POSIX error code that can be retrieved with `Tcl_GetErrno`.

**TCL_SEEK**

`Tcl_Seek` moves the access point in `channel` where subsequent data will be read or written. Buffered output is flushed to the channel and buffered input is discarded, prior to the seek operation.

`Tcl_Seek` normally returns the new access point. If an error occurs, `Tcl_Seek` returns −1 and records a POSIX error code that can be retrieved with `Tcl_GetErrno`. After an error, the access point may or may not have been moved.

**TCL_TELL**

`Tcl_Tell` returns the current access point for a channel. The returned value is −1 if the channel does not support seeking.
TCL_GETCHANNELOPTION

Tcl_GetChannelOption retrieves, in optionValue, the value of one of the options currently in effect for a channel, or a list of all options and their values. The channel argument identifies the channel for which to query an option or retrieve all options and their values. If optionName is not NULL, it is the name of the option to query; the option's value is copied to the Tcl dynamic string denoted by optionValue. If optionName is NULL, the function stores an alternating list of option names and their values in optionValue, using a series of calls to Tcl_DStringAppendElement. The various preexisting options and their possible values are described in the manual entry for the Tcl fconfigure command. Other options can be added by each channel type. These channel type specific options are described in the manual entry for the Tcl command that creates a channel of that type; for example, the additional options for TCP based channels are described in the manual entry for the Tcl socket command. The procedure normally returns TCL_OK. If an error occurs, it returns TCL_ERROR and calls Tcl_SetErrno to store an appropriate POSIX error code.

TCL_SETCHANNELOPTION

Tcl_SetChannelOption sets a new value newValue for an option optionName on channel. The procedure normally returns TCL_OK. If an error occurs, it returns TCL_ERROR; in addition, if interp is non−NULL, Tcl_SetChannelOption leaves an error message in the interpreter's result.

TCL_EOF

Tcl_Eof returns a nonzero value if channel encountered an end of file during the last input operation.

TCL_INPUTBLOCKED

Tcl_InputBlocked returns a nonzero value if channel is in nonblocking mode and the last input operation returned less data than requested because there was insufficient data available. The call always returns zero if the channel is in blocking mode.

TCL_INPUTBUFFERED

Tcl_InputBuffered returns the number of bytes of input currently buffered in the internal buffers for a channel. If the channel is not open for reading, this function always returns zero.

TCL_OUTPUTBUFFERED

Tcl_OutputBuffered returns the number of bytes of output currently buffered in the internal buffers for a channel. If the channel is not open for writing, this function always returns zero.

PLATFORM ISSUES

The handles returned from Tcl_GetChannelHandle depend on the platform and the channel type. On Unix platforms, the handle is always a Unix file descriptor as returned from the open system call. On Windows
platforms, the handle is a file HANDLE when the channel was created with Tcl_OpenFileChannel, Tcl_OpenCommandChannel, or Tcl_MakeFileChannel. Other channel types may return a different type of handle on Windows platforms. On the Macintosh platform, the handle is a file reference number as returned from HOpenDF.

SEE ALSO

DString, fconfigure, filename, fopen, Tcl_CreateChannel

KEYWORDS

access point, blocking, buffered I/O, channel, channel driver, end of file, flush, input, nonblocking, output, read, seek, write

Copyright ' 1996−1997 Sun Microsystems, Inc.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
NAME

Tcl_OpenTcpClient, Tcl_MakeTcpClientChannel, Tcl_OpenTcpServer − procedures to open channels using TCP sockets

SYNOPSIS

ARGUMENTS

DESCRIPTION

PLATFORM ISSUES

SEE ALSO

KEYWORDS
A string specifying the host name or address for network interface to use for the local end of the connection. If NULL, a default interface is chosen.

int async (in)

If nonzero, the client socket is connected asynchronously to the server.

ClientData sock (in)

Platform−specific handle for client TCP socket.

Tcl_TcpAcceptProc *proc (in)

Pointer to a procedure to invoke each time a new connection is accepted via the socket.

ClientData clientData (in)

Arbitrary one−word value to pass to proc.

**DESCRIPTION**

These functions are convenience procedures for creating channels that communicate over TCP sockets. The operations on a channel are described in the manual entry for *Tcl_OpenFileChannel*.

**TCL_OPENTCPCLIENT**

*Tcl_OpenTcpClient* opens a client TCP socket connected to a *port* on a specific *host*, and returns a channel that can be used to communicate with the server. The host to connect to can be specified either as a domain name style name (e.g. www.sunlabs.com), or as a string containing the alphanumeric representation of its four−byte address (e.g. 127.0.0.1). Use the string *localhost* to connect to a TCP socket on the host on which the function is invoked.

The *myaddr* and *myport* arguments allow a client to specify an address for the local end of the connection. If *myaddr* is NULL, then an interface is chosen automatically by the operating system. If *myport* is 0, then a port number is chosen at random by the operating system.

If *async* is zero, the call to *Tcl_OpenTcpClient* returns only after the client socket has either successfully connected to the server, or the attempted connection has failed. If *async* is nonzero the socket is connected asynchronously and the returned channel may not yet be connected to the server when the call to *Tcl_OpenTcpClient* returns. If the channel is in blocking mode and an input or output operation is done on the channel before the connection is completed or fails, that operation will wait until the connection either completes successfully or fails. If the channel is in nonblocking mode, the input or output operation will return immediately and a subsequent call to *Tcl_InputBlocked* on the channel will return nonzero.

The returned channel is opened for reading and writing. If an error occurs in opening the socket, *Tcl_OpenTcpClient* returns NULL and records a POSIX error code that can be retrieved with *Tcl_GetErrno*. In addition, if *interp* is non−NULL, an error message is left in the interpreter's result.

The newly created channel is not registered in the supplied interpreter; to register it, use *Tcl_RegisterChannel*. If one of the standard channels, stdin, stdout or stderr was previously closed, the act of creating the new channel also assigns it as a replacement for the standard channel.
TCL_MAKETCPCLIENTCHANNEL

Tcl_MakeTcpClientChannel creates a Tcl_Channel around an existing, platform specific, handle for a client TCP socket.

The newly created channel is not registered in the supplied interpreter; to register it, use Tcl_RegisterChannel. If one of the standard channels, stdin, stdout or stderr was previously closed, the act of creating the new channel also assigns it as a replacement for the standard channel.

TCL_OPENTCPSERVER

Tcl_OpenTcpServer opens a TCP socket on the local host on a specified port and uses the Tcl event mechanism to accept requests from clients to connect to it. The myaddr argument specifies the network interface. If myaddr is NULL the special address INADDR_ANY should be used to allow connections from any network interface. Each time a client connects to this socket, Tcl creates a channel for the new connection and invokes proc with information about the channel. Proc must match the following prototype:

```c
typedef void Tcl_TcpAcceptProc(
    ClientData clientData,
    Tcl_Channel channel,
    char *hostName,
    int port)
```

The clientData argument will be the same as the clientData argument to Tcl_OpenTcpServer, channel will be the handle for the new channel, hostName points to a string containing the name of the client host making the connection, and port will contain the client's port number. The new channel is opened for both input and output. If proc raises an error, the connection is closed automatically. Proc has no return value, but if it wishes to reject the connection it can close channel.

Tcl_OpenTcpServer normally returns a pointer to a channel representing the server socket. If an error occurs, Tcl_OpenTcpServer returns NULL and records a POSIX error code that can be retrieved with Tcl_GetErrno. In addition, if the interpreter is non−NULL, an error message is left in the interpreter's result.

The channel returned by Tcl_OpenTcpServer cannot be used for either input or output. It is simply a handle for the socket used to accept connections. The caller can close the channel to shut down the server and disallow further connections from new clients.

TCP server channels operate correctly only in applications that dispatch events through Tcl_DoOneEvent or through Tcl commands such as vwait; otherwise Tcl will never notice that a connection request from a remote client is pending.

The newly created channel is not registered in the supplied interpreter; to register it, use Tcl_RegisterChannel. If one of the standard channels, stdin, stdout or stderr was previously closed, the act of creating the new channel also assigns it as a replacement for the standard channel.
PLATFORM ISSUES

On Unix platforms, the socket handle is a Unix file descriptor as returned by the `socket` system call. On the Windows platform, the socket handle is a `SOCKET` as defined in the WinSock API. On the Macintosh platform, the socket handle is a `StreamPtr`.

SEE ALSO

`Tcl_OpenFileChannel`, `Tcl_RegisterChannel`, `vwait`

KEYWORDS

`client`, `server`, `TCP`
Panic

NAME

Tcl_Panic, Tcl_PanicVA, Tcl_SetPanicProc, panic, panicVA – report fatal error and abort

SYNOPSIS

#include <tcl.h>

void Tcl_Panic(const char* format, arg, arg, ...)
void Tcl_PanicVA(const char* format, va_list argList)
void Tcl_SetPanicProc(Tcl_PanicProc *panicProc)
void panic(const char* format, arg, arg, ...)
void panicVA(const char* format, va_list argList)

ARGUMENTS

CONST char* format (in)
    A printf–style format string.
arg (in)
    Arguments matching the format string.
va_list argList (in)
    An argument list of arguments matching the format string. Must have been initialized using
    TCL_VARARGS_START, and cleared using va_end.
Tcl_PanicProc *panicProc (in)
    Procedure to report fatal error message and abort.

DESCRIPTION

When the Tcl library detects that its internal data structures are in an inconsistent state, or that its C
procedures have been called in a manner inconsistent with their documentation, it calls Tcl_Panic to display a
message describing the error and abort the process. The format argument is a format string describing how to
format the remaining arguments arg into an error message, according to the same formatting rules used by the
printf family of functions. The same formatting rules are also used by the builtin Tcl command format.

In a freshly loaded Tcl library, Tcl_Panic prints the formatted error message to the standard error file of the
process, and then calls abort to terminate the process. Tcl_Panic does not return.
Tcl_SetPanicProc may be used to modify the behavior of Tcl_Panic. The panicProc argument should match the type Tcl_PanicProc:

```c
typedef void Tcl_PanicProc(
    const char *format,
    arg, arg, ...
);
```

After Tcl_SetPanicProc returns, any future calls to Tcl_Panic will call panicProc, passing along the format and arg arguments. To maintain consistency with the callers of Tcl_Panic, panicProc must not return; it must call abort. panicProc should avoid making calls into the Tcl library, or into other libraries that may call the Tcl library, since the original call to Tcl_Panic indicates the Tcl library is not in a state of reliable operation.

The typical use of Tcl_SetPanicProc arranges for the error message to be displayed or reported in a manner more suitable for the application or the platform. As an example, the Windows implementation of wish calls Tcl_SetPanicProc to force all panic messages to be displayed in a system dialog box, rather than to be printed to the standard error file (usually not visible under Windows).

Although the primary callers of Tcl_Panic are the procedures of the Tcl library, Tcl_Panic is a public function and may be called by any extension or application that wishes to abort the process and have a panic message displayed the same way that panic messages from Tcl will be displayed.

Tcl_PanicVA is the same as Tcl_Panic except that instead of taking a variable number of arguments it takes an argument list. The procedures panic and panicVA are synonyms (implemented as macros) for Tcl_Panic and Tcl_PanicVA, respectively. They exist to support old code; new code should use direct calls to Tcl_Panic or Tcl_PanicVA.

**SEE ALSO**

abort, printf, exec, format

**KEYWORDS**

abort, fatal, error

Copyright © 1995-1997 Roger E. Critchlow Jr.
ParseCmd

NAME

Tcl_ParseCommand, Tcl_ParseExpr, Tcl_ParseBraces, Tcl_ParseQuotedString, Tcl_ParseVarName, Tcl_ParseVar, Tcl_FreeParse, Tcl_EvalTokens, Tcl_EvalTokensStandard − parse Tcl scripts and expressions

SYNOPSIS

ARGUMENTS

DESCRIPTION

TCL_PARSE_STRUCTURE

TCL_TOKEN_WORD
TCL_TOKEN_SIMPLE_WORD
TCL_TOKEN_TEXT
TCL_TOKEN_BS
TCL_TOKEN_COMMAND
TCL_TOKEN_VARIABLE
TCL_TOKEN_SUB_EXPR
TCL_TOKEN_OPERATOR

KEYWORDS

NAME

Tcl_ParseCommand, Tcl_ParseExpr, Tcl_ParseBraces, Tcl_ParseQuotedString, Tcl_ParseVarName, Tcl_ParseVar, Tcl_FreeParse, Tcl_EvalTokens, Tcl_EvalTokensStandard − parse Tcl scripts and expressions

SYNOPSIS

#include <tcl.h>
int Tcl_ParseCommand(interp, string, numBytes, nested, parsePtr)
tclint Tcl_ParseExpr(interp, string, numBytes, parsePtr)
tclint Tcl_ParseBraces(interp, string, numBytes, parsePtr, append, termPtr)
tclint Tcl_ParseQuotedString(interp, string, numBytes, parsePtr, append, termPtr)
tclint Tcl_ParseVarName(interp, string, numBytes, parsePtr, append)
CONST char * Tcl_ParseVar(interp, string, termPtr)
tclint Tcl_FreeParse(usedParsePtr)
Tcl_Obj *
Tcl_EvalTokens(interp, tokenPtr, numTokens)

int
Tcl_EvalTokensStandard(interp, tokenPtr, numTokens)

ARGUMENTS

Tcl_Interp *interp (out)
For procedures other than Tcl_FreeParse, Tcl_EvalTokens and Tcl_EvalTokensStandard, used only for error reporting; if NULL, then no error messages are left after errors. For Tcl_EvalTokens and Tcl_EvalTokensStandard, determines the context for evaluating the script and also is used for error reporting; must not be NULL.

CONST char *string (in)
Pointer to first character in string to parse.

int numBytes (in)
Number of bytes in string, not including any terminating null character. If less than 0 then the script consists of all characters in string up to the first null character.

int nested (in)
Non−zero means that the script is part of a command substitution so an unquoted close bracket should be treated as a command terminator. If zero, close brackets have no special meaning.

int append (in)
Non−zero means that *parsePtr already contains valid tokens; the new tokens should be appended to those already present. Zero means that *parsePtr is uninitialized; any information in it is ignored. This argument is normally 0.

Tcl_Parse *parsePtr (out)
Points to structure to fill in with information about the parsed command, expression, variable name, etc. Any previous information in this structure is ignored, unless append is non−zero in a call to Tcl_ParseBraces, Tcl_ParseQuotedString, or Tcl_ParseVarName.

CONST char **termPtr (out)
If not NULL, points to a location where Tcl_ParseBraces, Tcl_ParseQuotedString, and Tcl_ParseVar will store a pointer to the character just after the terminating character (the close−brace, the last character of the variable name, or the close−quote (respectively)) if the parse was successful.

Tcl_Parse *usedParsePtr (in)
Points to structure that was filled in by a previous call to Tcl_ParseCommand, Tcl_ParseExpr, Tcl_ParseVarName, etc.

DESCRIPTION

These procedures parse Tcl commands or portions of Tcl commands such as expressions or references to variables. Each procedure takes a pointer to a script (or portion thereof) and fills in the structure pointed to by parsePtr with a collection of tokens describing the information that was parsed. The procedures normally return TCL_OK. However, if an error occurs then they return TCL_ERROR, leave an error message in interp's result (if interp is not NULL), and leave nothing in parsePtr.
Tcl_ParseCommand is a procedure that parses Tcl scripts. Given a pointer to a script, it parses the first command from the script. If the command was parsed successfully, Tcl_ParseCommand returns TCL_OK and fills in the structure pointed to by parsePtr with information about the structure of the command (see below for details). If an error occurred in parsing the command then TCL_ERROR is returned, an error message is left in interp's result, and no information is left at *parsePtr.

Tcl_ParseExpr parses Tcl expressions. Given a pointer to a script containing an expression, Tcl_ParseCommand parses the expression. If the expression was parsed successfully, Tcl_ParseExpr returns TCL_OK and fills in the structure pointed to by parsePtr with information about the structure of the expression (see below for details). If an error occurred in parsing the command then TCL_ERROR is returned, an error message is left in interp's result, and no information is left at *parsePtr.

Tcl_ParseBraces parses a string or command argument enclosed in braces such as {hello} or {string \t with \t tabs} from the beginning of its argument string. The first character of string must be {. If the braced string was parsed successfully, Tcl_ParseBraces returns TCL_OK, fills in the structure pointed to by parsePtr with information about the structure of the string (see below for details), and stores a pointer to the character just after the terminating } in the location given by *termPtr. If an error occurs while parsing the string then TCL_ERROR is returned, an error message is left in interp's result, and no information is left at *parsePtr or *termPtr.

Tcl_ParseQuotedString parses a double-quoted string such as "sum is [expr $a+$b]" from the beginning of the argument string. The first character of string must be ". If the double-quoted string was parsed successfully, Tcl_ParseQuotedString returns TCL_OK, fills in the structure pointed to by parsePtr with information about the structure of the string (see below for details), and stores a pointer to the character just after the terminating " in the location given by *termPtr. If an error occurs while parsing the string then TCL_ERROR is returned, an error message is left in interp's result, and no information is left at *parsePtr or *termPtr.

Tcl_ParseVarName parses a Tcl variable reference such as $abc or $x({expr $index + 1}) from the beginning of its string argument. The first character of string must be $. If a variable name was parsed successfully, Tcl_ParseVarName returns TCL_OK and fills in the structure pointed to by parsePtr with information about the structure of the variable name (see below for details). If an error occurs while parsing the command then TCL_ERROR is returned, an error message is left in interp's result (if interp isn't NULL), and no information is left at *parsePtr.

Tcl_ParseVar parse a Tcl variable reference such as $abc or $x({expr $index + 1}) from the beginning of its string argument. The first character of string must be $. If the variable name is parsed successfully, Tcl_ParseVar returns a pointer to the string value of the variable. If an error occurs while parsing, then NULL is returned and an error message is left in interp's result.

The information left at *parsePtr by Tcl_ParseCommand, Tcl_ParseExpr, Tcl_ParseBraces, Tcl_ParseQuotedString, and Tcl_ParseVarName may include dynamically allocated memory. If these five parsing procedures return TCL_OK then the caller must invoke Tcl_FreeParse to release the storage at *parsePtr. These procedures ignore any existing information in *parsePtr (unless append is non–zero), so if repeated calls are being made to any of them then Tcl_FreeParse must be invoked once after each call.
**Tcl_EvalTokensStandard** evaluates a sequence of parse tokens from a Tcl_Parse structure. The tokens typically consist of all the tokens in a word or all the tokens that make up the index for a reference to an array variable. **Tcl_EvalTokensStandard** performs the substitutions requested by the tokens and concatenates the resulting values. The return value from **Tcl_EvalTokensStandard** is a Tcl completion code with one of the values TCL_OK, TCL_ERROR, TCL_RETURN, TCL_BREAK, or TCL_CONTINUE, or possibly some other integer value originating in an extension. In addition, a result value or error message is left in interp's result; it can be retrieved using **Tcl_GetObjResult**.

**Tcl_EvalTokens** differs from **Tcl_EvalTokensStandard** only in the return convention used: it returns the result in a new Tcl_Obj. The reference count of the object returned as result has been incremented, so the caller must invoke **Tcl_DecrRefCount** when it is finished with the object. If an error or other exception occurs while evaluating the tokens (such as a reference to a non-existent variable) then the return value is NULL and an error message is left in interp's result. The use of **Tcl_EvalTokens** is deprecated.

**TCL_PARSE STRUCTURE**

**Tcl_ParseCommand**, **Tcl_ParseExpr**, **Tcl_ParseBraces**, **Tcl_ParseQuotedString**, and **Tcl_ParseVarName** return parse information in two data structures, Tcl_Parse and Tcl_Token:

```c
typedef struct Tcl_Parse {
    CONST char *commentStart;
    int commentSize;
    CONST char *commandStart;
    int commandSize;
    int numWords;
    Tcl_Token *tokenPtr;
    int numTokens;
    ...
} Tcl_Parse;

typedef struct Tcl_Token {
    int type;
    CONST char *start;
    int size;
    int numComponents;
} Tcl_Token;
```

The first five fields of a Tcl_Parse structure are filled in only by **Tcl_ParseCommand**. These fields are not used by the other parsing procedures.

**Tcl_ParseCommand** fills in a Tcl_Parse structure with information that describes one Tcl command and any comments that precede the command. If there are comments, the commentStart field points to the # character that begins the first comment and commentSize indicates the number of bytes in all of the comments preceding the command, including the newline character that terminates the last comment. If the command is not preceded by any comments, commentSize is 0. **Tcl_ParseCommand** also sets the commandStart field to point to the first character of the first word in the command (skipping any comments and leading space) and commandSize gives the total number of bytes in the command, including the character pointed to by commandStart up to and including the newline, close bracket, or semicolon character that terminates the
command. The \textit{numWords} field gives the total number of words in the command.

All parsing procedures set the remaining fields, \textit{tokenPtr} and \textit{numTokens}. The \textit{tokenPtr} field points to the first in an array of \texttt{Tcl-Token} structures that describe the components of the entity being parsed. The \textit{numTokens} field gives the total number of tokens present in the array. Each token contains four fields. The \textit{type} field selects one of several token types that are described below. The \textit{start} field points to the first character in the token and the \textit{size} field gives the total number of characters in the token. Some token types, such as \texttt{TCL_TOKEN_WORD} and \texttt{TCL_TOKEN_VARIABLE}, consist of several component tokens, which immediately follow the parent token; the \textit{numComponents} field describes how many of these there are. The \textit{type} field has one of the following values:

\texttt{TCL_TOKEN_WORD}

This token ordinarily describes one word of a command but it may also describe a quoted or braced string in an expression. The token describes a component of the script that is the result of concatenating together a sequence of subcomponents, each described by a separate subtoken. The token starts with the first non–blank character of the component (which may be a double–quote or open brace) and includes all characters in the component up to but not including the space, semicolon, close bracket, close quote, or close brace that terminates the component. The \textit{numComponents} field counts the total number of sub–tokens that make up the word, including sub–tokens of \texttt{TCL_TOKEN_VARIABLE} and \texttt{TCL_TOKEN_BS} tokens.

\texttt{TCL_TOKEN_SIMPLE_WORD}

This token has the same meaning as \texttt{TCL_TOKEN_WORD}, except that the word is guaranteed to consist of a single \texttt{TCL_TOKEN_TEXT} sub–token. The \textit{numComponents} field is always 1.

\texttt{TCL_TOKEN_TEXT}

The token describes a range of literal text that is part of a word. The \textit{numComponents} field is always 0.

\texttt{TCL_TOKEN_BS}

The token describes a backslash sequence such as \texttt{\textbackslash n} or \texttt{\textbackslash xa3}. The \textit{numComponents} field is always 0.

\texttt{TCL_TOKEN_COMMAND}

The token describes a command whose result result must be substituted into the word. The token includes the square brackets that surround the command. The \textit{numComponents} field is always 0 (the nested command is not parsed; call \texttt{Tcl_ParseCommand} recursively if you want to see its tokens).

\texttt{TCL_TOKEN_VARIABLE}

The token describes a variable substitution, including the $, variable name, and array index (if there is one) up through the close parenthesis that terminates the index. This token is followed by one or more additional tokens that describe the variable name and array index. If \textit{numComponents} is 1 then the variable is a scalar and the next token is a \texttt{TCL_TOKEN_TEXT} token that gives the variable name. If \textit{numComponents} is greater than 1 then the variable is an array: the first sub–token is a \texttt{TCL_TOKEN_TEXT} token giving the array name and the remaining sub–tokens are \texttt{TCL_TOKEN_TEXT}, \texttt{TCL_TOKEN_BS}, \texttt{TCL_TOKEN_COMMAND}, and \texttt{TCL_TOKEN_VARIABLE} tokens that must be concatenated to produce the array index. The \textit{numComponents} field includes nested sub–tokens that are part of \texttt{TCL_TOKEN_VARIABLE} tokens in the array index.

\texttt{TCL_TOKEN_SUB_EXPR}

The token describes one subexpression of an expression (or an entire expression). A subexpression
may consist of a value such as an integer literal, variable substitution, or parenthesized subexpression; it may also consist of an operator and its operands. The token starts with the first non-blank character of the subexpression up to but not including the space, brace, close-paren, or bracket that terminates the subexpression. This token is followed by one or more additional tokens that describe the subexpression. If the first sub-token after the `TCL_TOKEN_SUB_EXPR` token is a `TCL_TOKEN_OPERATOR` token, the subexpression consists of an operator and its token operands. If the operator has no operands, the subexpression consists of just the `TCL_TOKEN_OPERATOR` token. Each operand is described by a `TCL_TOKEN_SUB_EXPR` token. Otherwise, the subexpression is a value described by one of the token types `TCL_TOKEN_WORD`, `TCL_TOKEN_TEXT`, `TCL_TOKEN_COMMAND`, `TCL_TOKEN_VARIABLE`, and `TCL_TOKEN_SUB_EXPR`. The `numComponents` field counts the total number of sub-tokens that make up the subexpression; this includes the sub-tokens for any nested `TCL_TOKEN_SUB_EXPR` tokens.

**TCL_TOKEN_OPERATOR**

The token describes one operator of an expression such as `&&` or `hypot`. An `TCL_TOKEN_OPERATOR` token is always preceded by a `TCL_TOKEN_SUB_EXPR` token that describes the operator and its operands; the `TCL_TOKEN_SUB_EXPR` token's `numComponents` field can be used to determine the number of operands. A binary operator such as `*` is followed by two `TCL_TOKEN_SUB_EXPR` tokens that describe its operands. A unary operator like `−` is followed by a single `TCL_TOKEN_SUB_EXPR` token for its operand. If the operator is a math function such as `log10`, the `TCL_TOKEN_OPERATOR` token will give its name and the following `TCL_TOKEN_SUB_EXPR` tokens will describe its operands; if there are no operands (as with `rand`), no `TCL_TOKEN_SUB_EXPR` tokens follow. There is one trinary operator, `?`, that appears in `if−then−else` subexpressions such as `x ? y : z`; in this case, the `? TCL_TOKEN_OPERATOR` token is followed by three `TCL_TOKEN_SUB_EXPR` tokens for the operands `x`, `y`, and `z`. The `numComponents` field for a `TCL_TOKEN_OPERATOR` token is always 0.

After `Tcl_ParseCommand` returns, the first token pointed to by the `tokenPtr` field of the Tcl_Parse structure always has type `TCL_TOKEN_WORD` or `TCL_TOKEN_SIMPLE_WORD`. It is followed by the sub-tokens that must be concatenated to produce the value of that word. The next token is the `TCL_TOKEN_WORD` or `TCL_TOKEN_SIMPLE_WORD` token for the second word, followed by sub-tokens for that word, and so on until all `numWords` have been accounted for.

After `Tcl_ParseExpr` returns, the first token pointed to by the `tokenPtr` field of the Tcl_Parse structure always has type `TCL_TOKEN_SUB_EXPR`. It is followed by the sub-tokens that must be evaluated to produce the value of the expression. Only the token information in the Tcl_Parse structure is modified: the `commentStart`, `commentSize`, `commandStart`, and `commandSize` fields are not modified by `Tcl_ParseExpr`.

After `Tcl_ParseBraces` returns, the array of tokens pointed to by the `tokenPtr` field of the Tcl_Parse structure will contain a single `TCL_TOKEN_TEXT` token if the braced string does not contain any backslash–newlines. If the string does contain backslash–newlines, the array of tokens will contain one or more `TCL_TOKEN_TEXT` or `TCL_TOKEN_BS` sub-tokens that must be concatenated to produce the value of the string. If the braced string was just `{}` (that is, the string was empty), the single `TCL_TOKEN_TEXT` token will have a `size` field containing zero; this ensures that at least one token appears to describe the braced string. Only the token information in the Tcl_Parse structure is modified: the
commentStart, commentSize, commandStart, and commandSize fields are not modified by Tcl_ParseBraces.

After Tcl_ParseQuotedString returns, the array of tokens pointed to by the tokenPtr field of the Tcl_Parse structure depends on the contents of the quoted string. It will consist of one or more TCL_TOKEN_TEXT, TCL_TOKEN_BS, TCL_TOKEN_COMMAND, and TCL_TOKEN_VARIABLE sub–tokens. The array always contains at least one token; for example, if the argument string is empty, the array returned consists of a single TCL_TOKEN_TEXT token with a zero size field. Only the token information in the Tcl_Parse structure is modified: the commentStart, commentSize, commandStart, and commandSize fields are not modified.

After Tcl_ParseVarName returns, the first token pointed to by the tokenPtr field of the Tcl_Parse structure always has type TCL_TOKEN_VARIABLE. It is followed by the sub–tokens that make up the variable name as described above. The total length of the variable name is contained in the size field of the first token. As in Tcl_ParseExpr, only the token information in the Tcl_Parse structure is modified by Tcl_ParseVarName: the commentStart, commentSize, commandStart, and commandSize fields are not modified.

All of the character pointers in the Tcl_Parse and Tcl_Token structures refer to characters in the string argument passed to Tcl_ParseCommand, Tcl_ParseExpr, Tcl_ParseBraces, Tcl_ParseQuotedString, and Tcl_ParseVarName.

There are additional fields in the Tcl_Parse structure after the numTokens field, but these are for the private use of Tcl_ParseCommand, Tcl_ParseExpr, Tcl_ParseBraces, Tcl_ParseQuotedString, and Tcl_ParseVarName; they should not be referenced by code outside of these procedures.

KEYWORDS

backslash substitution, braces, command, expression, parse, token, variable substitution

Copyright ' 1997 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
PkgRequire

NAME

Tcl_PkgRequire, Tcl_PkgRequireEx, Tcl_PkgPresent, Tcl_PkgPresentEx, Tcl_PkgProvide,
Tcl_PkgProvideEx – package version control

SYNOPSIS

#include <tcl.h>
CONST char *
Tcl_PkgRequire(interp, name, version, exact)
CONST char *
Tcl_PkgRequireEx(interp, name, version, exact, clientDataPtr)
CONST char *
Tcl_PkgPresent(interp, name, version, exact)
CONST char *
Tcl_PkgPresentEx(interp, name, version, exact, clientDataPtr)
int
Tcl_PkgProvide(interp, name, version)
int
Tcl_PkgProvideEx(interp, name, version, clientData)

ARGUMENTS

Tcl_Interp *interp (in)
Interpreter where package is needed or available.
CONST char *name (in)
Name of package.
CONST char *version (in)
A version string consisting of one or more decimal numbers separated by dots.
int exact (in)
Non–zero means that only the particular version specified by version is acceptable. Zero means that
newer versions than version are also acceptable as long as they have the same major version number
as version.
ClientData clientData (in)
Arbitrary value to be associated with the package.
ClientData *clientDataPtr (out)
Pointer to place to store the value associated with the matching package. It is only changed if the
pointer is not NULL and the function completed successfully.
DESCRIPTION

These procedures provide C–level interfaces to Tk's package and version management facilities.

Tcl_PkgRequire is equivalent to the package require command, Tcl_PkgPresent is equivalent to the package present command, and Tcl_PkgProvide is equivalent to the package provide command.

See the documentation for the Tcl commands for details on what these procedures do.

If Tcl_PkgPresent or Tcl_PkgRequire complete successfully they return a pointer to the version string for the version of the package that is provided in the interpreter (which may be different than version); if an error occurs they return NULL and leave an error message in the interpreter's result.

Tcl_PkgProvide returns TCL_OK if it completes successfully; if an error occurs it returns TCL_ERROR and leaves an error message in the interpreter's result.

Tcl_PkgProvideEx, Tcl_PkgPresentEx and Tcl_PkgRequireEx allow the setting and retrieving of the client data associated with the package. In all other respects they are equivalent to the matching functions.

KEYWORDS

package, present, provide, require, version

Copyright ' 1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
Preserve

NAME

Tcl_Preserve, Tcl_Release, Tcl_EventuallyFree – avoid freeing storage while it's being used

SYNOPSIS

```
#include <tcl.h>
Tcl_Preserve(clientData)
Tcl_Release(clientData)
Tcl_EventuallyFree(clientData, freeProc)
```

ARGUMENTS

- **ClientData clientData (in)**
  
  Token describing structure to be freed or reallocated. Usually a pointer to memory for structure.

- **Tcl_FreeProc *freeProc (in)**
  
  Procedure to invoke to free clientData.

DESCRIPTION

These three procedures help implement a simple reference count mechanism for managing storage. They are designed to solve a problem having to do with widget deletion, but are also useful in many other situations. When a widget is deleted, its widget record (the structure holding information specific to the widget) must be returned to the storage allocator. However, it's possible that the widget record is in active use by one of the procedures on the stack at the time of the deletion. This can happen, for example, if the command associated with a button widget causes the button to be destroyed: an X event causes an event–handling C procedure in the button to be invoked, which in turn causes the button's associated Tcl command to be executed, which in turn causes the button to be deleted, which in turn causes the button's widget record to be de–allocated. Unfortunately, when the Tcl command returns, the button's event–handling procedure will need to reference the button's widget record. Because of this, the widget record must not be freed as part of the deletion, but must be retained until the event–handling procedure has finished with it. In other situations where the widget is deleted, it may be possible to free the widget record immediately.

**Tcl_Preserve** and **Tcl_Release** implement short–term reference counts for their **clientData** argument. The **clientData** argument identifies an object and usually consists of the address of a structure. The reference counts guarantee that an object will not be freed until each call to **Tcl_Preserve** for the object has been matched by calls to **Tcl_Release**. There may be any number of unmatched **Tcl_Preserve** calls in effect at once.

**Tcl_EventuallyFree** is invoked to free up its **clientData** argument. It checks to see if there are unmatched **Tcl_Preserve** calls for the object. If not, then **Tcl_EventuallyFree** calls **freeProc** immediately. Otherwise **Tcl_EventuallyFree** records the fact that **clientData** needs eventually to be freed. When all calls to
Tcl_Preserve have been matched with calls to Tcl_Release then freeProc will be called by Tcl_Release to do the cleanup.

All the work of freeing the object is carried out by freeProc. FreeProc must have arguments and result that match the type Tcl_FreeProc:

```c
typedef void Tcl_FreeProc(char *blockPtr);
```

The blockPtr argument to freeProc will be the same as the clientData argument to Tcl_EventuallyFree. The type of blockPtr (char *) is different than the type of the clientData argument to Tcl_EventuallyFree for historical reasons, but the value is the same.

When the clientData argument to Tcl_EventuallyFree refers to storage allocated and returned by a prior call to Tcl_Alloc, ealloc, or another function of the Tcl library, then the freeProc argument should be given the special value of TCL_DYNAMIC.

This mechanism can be used to solve the problem described above by placing Tcl_Preserve and Tcl_Release calls around actions that may cause undesired storage re-allocation. The mechanism is intended only for short-term use (i.e. while procedures are pending on the stack); it will not work efficiently as a mechanism for long-term reference counts. The implementation does not depend in any way on the internal structure of the objects being freed; it keeps the reference counts in a separate structure.

SEE ALSO

Tcl_Interp, Tcl_Alloc

KEYWORDS

free, reference count, storage

Copyright © 1990 The Regents of the University of California.  
Copyright © 1994-1996 Sun Microsystems, Inc.  
Copyright © 1995-1997 Roger E. Critchlow Jr.
PrintDbl

NAME

Tcl_PrintDouble – Convert floating value to string

SYNOPSIS

#include <tcl.h>
Tcl_PrintDouble(interp, value, dst)

ARGUMENTS

_Tcl_Interp *interp (in)
Before Tcl 8.0, the tcl_precision variable in this interpreter controlled the conversion. As of Tcl 8.0, this argument is ignored and the conversion is controlled by the tcl_precision variable that is now shared by all interpreters.

double value (in)
Floating-point value to be converted.

char *dst (out)
Where to store string representing value. Must have at least TCL_DOUBLE_SPACE characters of storage.

DESCRIPTION

Tcl_PrintDouble generates a string that represents the value of value and stores it in memory at the location given by dst. It uses %g format to generate the string, with one special twist: the string is guaranteed to contain either a `.0' or an `e' so that it doesn't look like an integer. Where %g would generate an integer with no decimal point, Tcl_PrintDouble adds `.0'.
Environment

NAME

Tcl_PutEnv – procedures to manipulate the environment

SYNOPSIS

#include <tcl.h>
int
Tcl_PutEnv(string)

ARGUMENTS

CONST char *string (in)

Info about environment variable in the form NAME=value. The string is in native format.

DESCRIPTION

Tcl_PutEnv sets an environment variable. The information is passed in a single string of the form NAME=value. This procedure is intended to be a stand–in for the UNIX putenv system call. All tcl–based applications using putenv should redefine it to Tcl_PutEnv so that they will interface properly to the Tcl runtime.

KEYWORDS

environment, variable
RecordEval

NAME

Tcl_RecordAndEval – save command on history list before evaluating

SYNOPSIS

#include <tcl.h>

int Tcl_RecordAndEval(interp, cmd, flags)

ARGUMENTS

Tcl_Interp *interp (in)
    Tcl interpreter in which to evaluate command.
CONST char *cmd (in)
    Command (or sequence of commands) to execute.
int flags (in)
    An OR'ed combination of flag bits. TCL_NO_EVAL means record the command but don't evaluate it.
    TCL_EVAL_GLOBAL means evaluate the command at global level instead of the current stack level.

DESCRIPTION

Tcl_RecordAndEval is invoked to record a command as an event on the history list and then execute it using Tcl_Eval (or Tcl_GlobalEval if the TCL_EVAL_GLOBAL bit is set in flags). It returns a completion code such as TCL_OK just like Tcl_Eval and it leaves information in the interpreter's result. If you don't want the command recorded on the history list then you should invoke Tcl_Eval instead of Tcl_RecordAndEval. Normally Tcl_RecordAndEval is only called with top−level commands typed by the user, since the purpose of history is to allow the user to re−issue recently−invoked commands. If the flags argument contains the TCL_NO_EVAL bit then the command is recorded without being evaluated.

Note that Tcl_RecordAndEval has been largely replaced by the object−based procedure Tcl_RecordAndEvalObj. That object−based procedure records and optionally executes a command held in a Tcl object instead of a string.

SEE ALSO

Tcl_RecordAndEvalObj
command, event, execute, history, interpreter, record

Copyright ' 1989-1993 The Regents of the University of California.
Copyright ' 1994-1997 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
RecEvalObj

NAME

Tcl_RecordAndEvalObj − save command on history list before evaluating

SYNOPSIS

#include <tcl.h>

int Tcl_RecordAndEvalObj(interp, cmdPtr, flags)

ARGUMENTS

  Tcl_Interp *interp (in)
    Tcl interpreter in which to evaluate command.
  Tcl_Obj *cmdPtr (in)
    Points to a Tcl object containing a command (or sequence of commands) to execute.
  int flags (in)
    An OR'ed combination of flag bits. TCL_NO_EVAL means record the command but don't evaluate it.
    TCL_EVAL_GLOBAL means evaluate the command at global level instead of the current stack level.

DESCRIPTION

Tcl_RecordAndEvalObj is invoked to record a command as an event on the history list and then execute it using Tcl_EvalObjEx (or Tcl_GlobalEvalObj if the TCL_EVAL_GLOBAL bit is set in flags). It returns a completion code such as TCL_OK just like Tcl_EvalObjEx, as well as a result object containing additional information (a result value or error message) that can be retrieved using Tcl_GetObjResult. If you don't want the command recorded on the history list then you should invoke Tcl_EvalObjEx instead of Tcl_RecordAndEvalObj. Normally Tcl_RecordAndEvalObj is only called with top−level commands typed by the user, since the purpose of history is to allow the user to re−issue recently−invoked commands. If the flags argument contains the TCL_NO_EVAL bit then the command is recorded without being evaluated.

SEE ALSO

Tcl_EvalObjEx, Tcl_GetObjResult

KEYWORDS

command, event, execute, history, interpreter, object, record

Copyright ' 1997 Sun Microsystems, Inc.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
RegExp

NAME

Tcl_RegExpMatch, Tcl_RegExpCompile, Tcl_RegExpExec, Tcl_RegExpRange, Tcl_GetRegExpFromObj, Tcl_RegExpMatchObj, Tcl_RegExpExecObj, Tcl_RegExpGetInfo – Pattern matching with regular expressions

SYNOPSIS

ARGUMENTS

DESCRIPTION

TCL_REG_ADVANCED
TCL_REG_EXTENDED
TCL_REG_BASIC
TCL_REG_EXPANDED
TCL_REG_QUOTE
TCL_REG_NOCASE
TCL_REG_NEWLINE
TCL_REG_NLSTOP
TCL_REG_NLANCH
TCL_REG_NOSUB
TCL_REG_CANMATCH
TCL_REG_NOTBOL
TCL_REG_NOTEOL

SEE ALSO

KEYWORDS

NAME

Tcl_RegExpMatch, Tcl_RegExpCompile, Tcl_RegExpExec, Tcl_RegExpRange, Tcl_GetRegExpFromObj, Tcl_RegExpMatchObj, Tcl_RegExpExecObj, Tcl_RegExpGetInfo – Pattern matching with regular expressions

SYNOPSIS

#include <tcl.h>

int
Tcl_RegExpMatchObj(interp, strObj, patObj)

int
Tcl_RegExpMatch(interp, string, pattern)

Tcl_RegExp

Tcl_RegExpCompile(interp, pattern)

int
Tcl_RegExpExec(interp, regexp, string, start)
Tcl_RegExpRange(regexp, index, startPtr, endPtr)
Tcl_RegExp
Tcl_GetRegExpFromObj(interp, patObj, cflags)
int
Tcl_RegExpExecObj(interp, regexp, objPtr, offset, nmatches, eflags)
Tcl_RegExpGetInfo(regexp, infoPtr)

ARGUMENTS

Tcl_Interp *interp (in)
Tcl interpreter to use for error reporting. The interpreter may be NULL if no error reporting is desired.

Tcl_Obj *strObj (in/out)
Refers to the object from which to get the string to search. The internal representation of the object may be converted to a form that can be efficiently searched.

Tcl_Obj *patObj (in/out)
Refers to the object from which to get a regular expression. The compiled regular expression is cached in the object.

char *string (in)
String to check for a match with a regular expression.

CONST char *pattern (in)
String in the form of a regular expression pattern.

Tcl_RegExp regexp (in)
Compiled regular expression. Must have been returned previously by Tcl_GetRegExpFromObj or Tcl_RegExpCompile.

char *start (in)
If string is just a portion of some other string, this argument identifies the beginning of the larger string. If it isn't the same as string, then no ^ matches will be allowed.

int index (in)
Specifies which range is desired: 0 means the range of the entire match, 1 or greater means the range that matched a parenthesized sub–expression.

CONST char **startPtr (out)
The address of the first character in the range is stored here, or NULL if there is no such range.

CONST char **endPtr (out)
The address of the character just after the last one in the range is stored here, or NULL if there is no such range.

int cflags (in)
OR–ed combination of compilation flags. See below for more information.

Tcl_Obj *objPtr (in/out)
An object which contains the string to check for a match with a regular expression.

int offset (in)
The character offset into the string where matching should begin. The value of the offset has no impact on ^ matches. This behavior is controlled by eflags.

int nmatches (in)
The number of matching subexpressions that should be remembered for later use. If this value is 0,
then no subexpression match information will be computed. If the value is −1, then all of the
matching subexpressions will be remembered. Any other value will be taken as the maximum number
of subexpressions to remember.

\textbf{int eflags (in)}

\begin{itemize}
  \item OR–ed combination of the values TCL_REG_NOTBOL and TCL_REG_NOTEOL. See below for
        more information.
\end{itemize}

\textbf{Tcl\_RegExpInfo *infoPtr (out)}

The address of the location where information about a previous match should be stored by
\textbf{Tcl\_RegExpGetInfo}.

\section*{DESCRIPTION}

\textbf{Tcl\_RegExpMatch} determines whether its \texttt{pattern} argument matches \texttt{regexp}, where \texttt{regexp} is interpreted as a
regular expression using the rules in the \texttt{re\_syntax} reference page. If there is a match then
\textbf{Tcl\_RegExpMatch} returns \texttt{1}. If there is no match then \textbf{Tcl\_RegExpMatch} returns \texttt{0}. If an error occurs in the
matching process (e.g. \texttt{pattern} is not a valid regular expression) then \textbf{Tcl\_RegExpMatch} returns \texttt{−1} and
leaves an error message in the interpreter result. \textbf{Tcl\_RegExpMatchObj} is similar to \textbf{Tcl\_RegExpMatch}
except it operates on the Tcl objects \texttt{strObj} and \texttt{patObj} instead of UTF strings. \textbf{Tcl\_RegExpMatchObj} is
generally more efficient than \textbf{Tcl\_RegExpMatch}, so it is the preferred interface.

\textbf{Tcl\_RegExpCompile}, \textbf{Tcl\_RegExpExec}, and \textbf{Tcl\_RegExpRange} provide lower–level access to the regular
expression pattern matcher. \textbf{Tcl\_RegExpCompile} compiles a regular expression string into the internal form
used for efficient pattern matching. The return value is a token for this compiled form, which can be used in
subsequent calls to \textbf{Tcl\_RegExpExec} or \textbf{Tcl\_RegExpRange}. If an error occurs while compiling the regular
expression then \textbf{Tcl\_RegExpCompile} returns NULL and leaves an error message in the interpreter result.
Note: the return value from \textbf{Tcl\_RegExpCompile} is only valid up to the next call to \textbf{Tcl\_RegExpCompile}; it
is not safe to retain these values for long periods of time.

\textbf{Tcl\_RegExpExec} executes the regular expression pattern matcher. It returns \texttt{1} if \texttt{string} contains a range of
characters that match \texttt{regexp}, \texttt{0} if no match is found, and \texttt{−1} if an error occurs. In the case of an error,
\textbf{Tcl\_RegExpExec} leaves an error message in the interpreter result. When searching a string for multiple
matches of a pattern, it is important to distinguish between the start of the original string and the start of the
current search. For example, when searching for the second occurrence of a match, the \texttt{string} argument might
point to the character just after the first match; however, it is important for the pattern matcher to know that
this is not the start of the entire string, so that it doesn't allow \texttt{^} atoms in the pattern to match. The \texttt{start}
argument provides this information by pointing to the start of the overall string containing \texttt{string}. \texttt{Start} will be
less than or equal to \texttt{string}; if it is less than \texttt{string} then no \texttt{^} matches will be allowed.

\textbf{Tcl\_RegExpRange} may be invoked after \textbf{Tcl\_RegExpExec} returns; it provides detailed information about
what ranges of the string matched what parts of the pattern. \textbf{Tcl\_RegExpRange} returns a pair of pointers in
*\texttt{startPtr} and *\texttt{endPtr} that identify a range of characters in the source string for the most recent call to
\textbf{Tcl\_RegExpExec}. \texttt{Index} indicates which of several ranges is desired: if \texttt{index} is \texttt{0}, information is returned
about the overall range of characters that matched the entire pattern; otherwise, information is returned about
the range of characters that matched the \texttt{index}th parenthesized subexpression within the pattern. If there is no
range corresponding to \texttt{index} then NULL is stored in *\texttt{startPtr} and *\texttt{endPtr}.
Tcl_GetRegExpFromObj, Tcl_RegExpExecObj, and Tcl_RegExpGetInfo are object interfaces that provide the most direct control of Henry Spencer's regular expression library. For users that need to modify compilation and execution options directly, it is recommended that you use these interfaces instead of calling the internal regexp functions. These interfaces handle the details of UTF to Unicode translations as well as providing improved performance through caching in the pattern and string objects.

Tcl_GetRegExpFromObj attempts to return a compiled regular expression from the patObj. If the object does not already contain a compiled regular expression it will attempt to create one from the string in the object and assign it to the internal representation of the patObj. The return value of this function is of type Tcl_RegExp. The return value is a token for this compiled form, which can be used in subsequent calls to Tcl_RegExpExecObj or Tcl_RegExpGetInfo. If an error occurs while compiling the regular expression then Tcl_GetRegExpFromObj returns NULL and leaves an error message in the interpreter result. The regular expression token can be used as long as the internal representation of patObj refers to the compiled form. The eflags argument is a bitwise OR of zero or more of the following flags that control the compilation of patObj:

TCL_REG_ADVANCED
Compile advanced regular expressions (`AREs'). This mode corresponds to the normal regular expression syntax accepted by the Tcl regexp and regsub commands.

TCL_REG_EXTENDED
Compile extended regular expressions (`EREs'). This mode corresponds to the regular expression syntax recognized by Tcl 8.0 and earlier versions.

TCL_REG_BASIC
Compile basic regular expressions (`BREs'). This mode corresponds to the regular expression syntax recognized by common Unix utilities like sed and grep. This is the default if no flags are specified.

TCL_REG_EXPANDED
Compile the regular expression (basic, extended, or advanced) using an expanded syntax that allows comments and whitespace. This mode causes non–backslashed non–bracket–expression white space and #–to–end–of–line comments to be ignored.

TCL_REG_QUOTE
Compile a literal string, with all characters treated as ordinary characters.

TCL_REG_NOCASE
Compile for matching that ignores upper/lower case distinctions.

TCL_REG_NEWLINE
Compile for newline–sensitive matching. By default, newline is a completely ordinary character with no special meaning in either regular expressions or strings. With this flag, `[^' bracket expressions and `. never match newline, `^' matches an empty string after any newline in addition to its normal function, and `$' matches an empty string before any newline in addition to its normal function. REG_NEWLINE is the bitwise OR of REG_NLSTOP and REG_NLANCH.

TCL_REG_NLSTOP
Compile for partial newline–sensitive matching, with the behavior of `[^' bracket expressions and `. affected, but not the behavior of `^' and `$'. In this mode, `[^' bracket expressions and `. never match newline.

TCL_REG_NLANCH
Compile for inverse partial newline–sensitive matching, with the behavior of of `^' and `$' (the `anchors") affected, but not the behavior of `[^' bracket expressions and `. In this mode `^' matches
an empty string after any newline in addition to its normal function, and `'$' matches an empty string before any newline in addition to its normal function.

**TCL_REG_NOSUB**
Compile for matching that reports only success or failure, not what was matched. This reduces compile overhead and may improve performance. Subsequent calls to `Tcl_RegExpGetInfo` or `Tcl_RegExpRange` will not report any match information.

**TCL_REG_CANMATCH**
Compile for matching that reports the potential to complete a partial match given more text (see below).

Only one of TCL_REG_EXTENDED, TCL_REG_ADVANCED, TCL_REG_BASIC, and TCL_REG_QUOTE may be specified.

`Tcl_RegExpExecObj` executes the regular expression pattern matcher. It returns 1 if `objPtr` contains a range of characters that match `regexp`, 0 if no match is found, and −1 if an error occurs. In the case of an error, `Tcl_RegExpExecObj` leaves an error message in the interpreter result. The `nmatches` value indicates to the matcher how many subexpressions are of interest. If `nmatches` is 0, then no subexpression match information is recorded, which may allow the matcher to make various optimizations. If the value is −1, then all of the subexpressions in the pattern are remembered. If the value is a positive integer, then only that number of subexpressions will be remembered. Matching begins at the specified Unicode character index given by `offset`. Unlike `Tcl_RegExpExec`, the behavior of anchors is not affected by the `offset` value. Instead the behavior of the anchors is explicitly controlled by the `eflags` argument, which is a bitwise OR of zero or more of the following flags:

**TCL_REG_NOTBOL**
The starting character will not be treated as the beginning of a line or the beginning of the string, so `'^'` will not match there. Note that this flag has no effect on how `\A` matches.

**TCL_REG_NOTEOL**
The last character in the string will not be treated as the end of a line or the end of the string, so `'$'` will not match there. Note that this flag has no effect on how `\Z` matches.

`Tcl_RegExpGetInfo` retrieves information about the last match performed with a given regular expression `regexp`. The `infoPtr` argument contains a pointer to a structure that is defined as follows:

```c
typedef struct Tcl_RegExpInfo {
    int nsubs;
    Tcl_RegExpIndices *matches;
    long extendStart;
} Tcl_RegExpInfo;
```

The `nsubs` field contains a count of the number of parenthesized subexpressions within the regular expression. If the TCL_REG_NOSUB was used, then this value will be zero. The `matches` field points to an array of `nsubs` values that indicate the bounds of each subexpression matched. The first element in the array refers to the range matched by the entire regular expression, and subsequent elements refer to the parenthesized subexpressions in the order that they appear in the pattern. Each element is a structure that is defined as follows:
typedef struct Tcl_RegExpIndices {
    long start;
    long end;
} Tcl_RegExpIndices;

The `start` and `end` values are Unicode character indices relative to the offset location within `objPtr` where matching began. The `start` index identifies the first character of the matched subexpression. The `end` index identifies the first character after the matched subexpression. If the subexpression matched the empty string, then `start` and `end` will be equal. If the subexpression did not participate in the match, then `start` and `end` will be set to −1.

The `extendStart` field in `Tcl_RegExpInfo` is only set if the `TCL_REG_CANMATCH` flag was used. It indicates the first character in the string where a match could occur. If a match was found, this will be the same as the beginning of the current match. If no match was found, then it indicates the earliest point at which a match might occur if additional text is appended to the string. If it is no match is possible even with further text, this field will be set to −1.

**SEE ALSO**

`re_syntax`

**KEYWORDS**

match, pattern, regular expression, string, subexpression, `Tcl_RegExpIndices`, `Tcl_RegExpInfo`
SaveResult

NAME

Tcl_SaveResult, Tcl_RestoreResult, Tcl_DiscardResult – save and restore an interpreter's result

SYNOPSIS

#include <tcl.h>
Tcl_SaveResult(interp, statePtr)
Tcl_RestoreResult(interp, statePtr)
Tcl_DiscardResult(statePtr)

ARGUMENTS

Tcl_Interp *interp (in)
Interpreter for which state should be saved.
Tcl_SavedResult *statePtr (in)
Pointer to location where interpreter result should be saved or restored.

DESCRIPTION

These routines allows a C procedure to take a snapshot of the current interpreter result so that it can be restored after a call to Tcl_Eval or some other routine that modifies the interpreter result. These routines are passed a pointer to a structure that is used to store enough information to restore the interpreter result state. This structure can be allocated on the stack of the calling procedure. These routines do not save the state of any error information in the interpreter (e.g. the errorCode or errorInfo variables).

Tcl_SaveResult moves the string and object results of interp into the location specified by statePtr. Tcl_SaveResult clears the result for interp and leaves the result in its normal empty initialized state.

Tcl_RestoreResult moves the string and object results from statePtr back into interp. Any result or error that was already in the interpreter will be cleared. The statePtr is left in an uninitialized state and cannot be used until another call to Tcl_SaveResult.

Tcl_DiscardResult releases the saved interpreter state stored at statePtr. The state structure is left in an uninitialized state and cannot be used until another call to Tcl_SaveResult.

Once Tcl_SaveResult is called to save the interpreter result, either Tcl_RestoreResult or Tcl_DiscardResult must be called to properly clean up the memory associated with the saved state.
KEYWORDS

result, state, interp

Copyright ' 1997 by Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
AssocData

NAME

Tcl_GetAssocData, Tcl_SetAssocData, Tcl_DeleteAssocData − manage associations of string keys and user specified data with Tcl interpreters.

SYNOPSIS

#include <tcl.h>

ClientData

Tcl_GetAssocData(interp, key, delProcPtr)

Tcl_SetAssocData(interp, key, delProc, clientData)

Tcl_DeleteAssocData(interp, key)

ARGUMENTS

Tcl_Interp *interp (in)

Interpreter in which to execute the specified command.

CONST char *key (in)

Key for association with which to store data or from which to delete or retrieve data. Typically the module prefix for a package.

Tcl_InterpDeleteProc *delProc (in)

Procedure to call when interp is deleted.

Tcl_InterpDeleteProc **delProcPtr (in)

Pointer to location in which to store address of current deletion procedure for association. Ignored if NULL.

ClientData clientData (in)

Arbitrary one-word value associated with the given key in this interpreter. This data is owned by the caller.

DESCRIPTION

These procedures allow extensions to associate their own data with a Tcl interpreter. An association consists of a string key, typically the name of the extension, and a one-word value, which is typically a pointer to a data structure holding data specific to the extension. Tcl makes no interpretation of either the key or the value for an association.

Storage management is facilitated by storing with each association a procedure to call when the interpreter is deleted. This procedure can dispose of the storage occupied by the client's data in any way it sees fit.

Tcl_SetAssocData creates an association between a string key and a user specified datum in the given interpreter. If there is already an association with the given key, Tcl_SetAssocData overwrites it with the new information. It is up to callers to organize their use of names to avoid conflicts, for example, by using package

AssocData
names as the keys. If the `deleteProc` argument is non-NULL it specifies the address of a procedure to invoke if the interpreter is deleted before the association is deleted. `DeleteProc` should have arguments and result that match the type `Tcl_InterpDeleteProc`:

```c
typedef void Tcl_InterpDeleteProc(
    ClientData clientData,
    Tcl_Interp  * interp);
```

When `deleteProc` is invoked the `clientData` and `interp` arguments will be the same as the corresponding arguments passed to `Tcl_SetAssocData`. The deletion procedure will not be invoked if the association is deleted before the interpreter is deleted.

`Tcl_GetAssocData` returns the datum stored in the association with the specified key in the given interpreter, and if the `delProcPtr` field is non-NULL, the address indicated by it gets the address of the delete procedure stored with this association. If no association with the specified key exists in the given interpreter `Tcl_GetAssocData` returns NULL.

`Tcl_DeleteAssocData` deletes an association with a specified key in the given interpreter. Then it calls the deletion procedure.

**KEYWORDS**

association, data, deletion procedure, interpreter, key
SetErrno

NAME

Tcl_SetErrno, Tcl_GetErrno, Tcl_ErrnoId, Tcl_ErrnoMsg – manipulate errno to store and retrieve error codes

SYNOPSIS

#include <tcl.h>

void
Tcl_SetErrno(errorCode)

int
Tcl_GetErrno()

CONST char *
Tcl_ErrnoId()

CONST char *
Tcl_ErrnoMsg(errorCode)

ARGUMENTS

int errorCode (in)
    A POSIX error code such as ENOENT.

DESCRIPTION

Tcl_SetErrno and Tcl_GetErrno provide portable access to the errno variable, which is used to record a POSIX error code after system calls and other operations such as Tcl_Gets. These procedures are necessary because global variable accesses cannot be made across module boundaries on some platforms.

Tcl_SetErrno sets the errno variable to the value of the errorCode argument C procedures that wish to return error information to their callers via errno should call Tcl_SetErrno rather than setting errno directly.

Tcl_GetErrno returns the current value of errno. Procedures wishing to access errno should call this procedure instead of accessing errno directly.

Tcl_ErrnoId and Tcl_ErrnoMsg return string representations of errno values. Tcl_ErrnoId returns a machine-readable textual identifier such as "EACCES" that corresponds to the current value of errno. Tcl_ErrnoMsg returns a human-readable string such as "permission denied" that corresponds to the value of its errorCode argument. The errorCode argument is typically the value returned by Tcl_GetErrno. The strings returned by these functions are statically allocated and the caller must not free or modify them.
KEYWORDS

erro, error code, global variables

Copyright © 1996 Sun Microsystems, Inc.
Copyright © 1995−1997 Roger E. Critchlow Jr.
SetRecLmt

NAME

Tcl_SetRecursionLimit – set maximum allowable nesting depth in interpreter

SYNOPSIS

#include <tcl.h>
int
Tcl_SetRecursionLimit(interp, depth)

ARGUMENTS

Tcl_Interp *interp (in)
    Interpreter whose recursion limit is to be set. Must be greater than zero.
int depth (in)
    New limit for nested calls to Tcl_Eval for interp.

DESCRIPTION

At any given time Tcl enforces a limit on the number of recursive calls that may be active for Tcl_Eval and related procedures such as Tcl_GlobalEval. Any call to Tcl_Eval that exceeds this depth is aborted with an error. By default the recursion limit is 1000.

Tcl_SetRecursionLimit may be used to change the maximum allowable nesting depth for an interpreter. The depth argument specifies a new limit for interp, and Tcl_SetRecursionLimit returns the old limit. To read out the old limit without modifying it, invoke Tcl_SetRecursionLimit with depth equal to 0.

The Tcl_SetRecursionLimit only sets the size of the Tcl call stack: it cannot by itself prevent stack overflows on the C stack being used by the application. If your machine has a limit on the size of the C stack, you may get stack overflows before reaching the limit set by Tcl_SetRecursionLimit. If this happens, see if there is a mechanism in your system for increasing the maximum size of the C stack.

KEYWORDS

nesting depth, recursion

Copyright ' 1989-1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
SetResult

NAME

Tcl_SetObjResult, Tcl_GetObjResult, Tcl_SetResult, Tcl_GetStringResult, Tcl_AppendResult,
Tcl_AppendResultVA, Tcl_AppendElement, Tcl_ResetResult, Tcl_FreeResult – manipulate Tcl result

SYNOPSIS

#include <tcl.h>

Tcl_SetObjResult(interp, objPtr)
Tcl_Obj *
Tcl_GetObjResult(interp)
Tcl_SetResult(interp, string, freeProc)
CONST char *
Tcl_GetStringResult(interp)
Tcl_AppendResult(interp, string, string, ... , (char *) NULL)
Tcl_AppendResultVA(interp, argList)
Tcl_AppendElement(interp, string)
Tcl_ResetResult(interp)
Tcl_FreeResult(interp)

ARGUMENTS

Tcl_Interp *interp (out)
Interpreter whose result is to be modified or read.
Tcl_Obj *objPtr (in)
Object value to become result for interp.
char *string (in)
String value to become result for interp or to be appended to the existing result.
Tcl_FreeProc *freeProc (in)
Address of procedure to call to release storage at string, or TCL_STATIC, TCL_DYNAMIC, or
TCL_VOLATILE.
va_list argList (in)
An argument list which must have been initialised using TCL_VARARGS_START, and cleared
using va_end.

DESCRIPTION

The procedures described here are utilities for manipulating the result value in a Tcl interpreter. The
interpreter result may be either a Tcl object or a string. For example, Tcl_SetObjResult and Tcl_SetResult
set the interpreter result to, respectively, an object and a string. Similarly, Tcl_GetObjResult and
Tcl_GetStringResult return the interpreter result as an object and as a string. The procedures always keep the
string and object forms of the interpreter result consistent. For example, if Tcl_SetObjResult is called to set
the result to an object, then **Tcl_GetStringResult** is called, it will return the object's string value.

**Tcl_SetObjResult** arranges for *objPtr* to be the result for *interp*, replacing any existing result. The result is left pointing to the object referenced by *objPtr*. *objPtr*'s reference count is incremented since there is now a new reference to it from *interp*. The reference count for any old result object is decremented and the old result object is freed if no references to it remain.

**Tcl_GetObjResult** returns the result for *interp* as an object. The object's reference count is not incremented; if the caller needs to retain a long−term pointer to the object they should use **Tcl_IncrRefCount** to increment its reference count in order to keep it from being freed too early or accidently changed.

**Tcl_SetResult** arranges for *string* to be the result for the current Tcl command in *interp*, replacing any existing result. The *freeProc* argument specifies how to manage the storage for the *string* argument; it is discussed in the section **THE TCL_FREEPROC ARGUMENT TO TCL_SETRESULT** below. If *string* is **NULL**, then *freeProc* is ignored and **Tcl_SetResult** re−initializes *interp*'s result to point to an empty string.

**Tcl_GetStringResult** returns the result for *interp* as an string. If the result was set to an object by a **Tcl_SetObjResult** call, the object form will be converted to a string and returned. If the object's string representation contains null bytes, this conversion will lose information. For this reason, programmers are encouraged to write their code to use the new object API procedures and to call **Tcl_GetObjResult** instead.

**Tcl_ResetResult** clears the result for *interp* and leaves the result in its normal empty initialized state. If the result is an object, its reference count is decremented and the result is left pointing to an unshared object representing an empty string. If the result is a dynamically allocated string, its memory is free*d* and the result is left as a empty string. **Tcl_ResetResult** also clears the error state managed by **Tcl_AddErrorInfo**, **Tcl_AddObjErrorInfo**, and **Tcl_SetErrorCode**.

**OLD STRING PROCEEDURES**

Use of the following procedures is deprecated since they manipulate the Tcl result as a string. Procedures such as **Tcl_SetObjResult** that manipulate the result as an object can be significantly more efficient.

**Tcl_AppendResult** makes it easy to build up Tcl results in pieces. It takes each of its *string* arguments and appends them in order to the current result associated with *interp*. If the result is in its initialized empty state (e.g. a command procedure was just invoked or **Tcl_ResetResult** was just called), then **Tcl_AppendResult** sets the result to the concatenation of its *string* arguments. **Tcl_AppendResult** may be called repeatedly as additional pieces of the result are produced. **Tcl_AppendResult** takes care of all the storage management issues associated with managing *interp*'s result, such as allocating a larger result area if necessary. It also converts the current interpreter result from an object to a string, if necessary, before appending the argument strings. Any number of *string* arguments may be passed in a single call; the last argument in the list must be a **NULL** pointer.

**Tcl_AppendResultVA** is the same as **Tcl_AppendResult** except that instead of taking a variable number of arguments it takes an argument list.
**Tcl_AppendElement** is similar to **Tcl_AppendResult** in that it allows results to be built up in pieces. However, **Tcl_AppendElement** takes only a single *string* argument and it appends that argument to the current result as a proper Tcl list element. **Tcl_AppendElement** adds backslashes or braces if necessary to ensure that *interp*'s result can be parsed as a list and that *string* will be extracted as a single element. Under normal conditions, **Tcl_AppendElement** will add a space character to *interp*'s result just before adding the new list element, so that the list elements in the result are properly separated. However if the new list element is the first in a list or sub−list (i.e. *interp*'s current result is empty, or consists of the single character `"{"`, or ends in the characters `"{"`) then no space is added.

**Tcl_FreeResult** performs part of the work of **Tcl_ResetResult**. It frees up the memory associated with *interp*'s result. It also sets *interp−>freeProc* to zero, but doesn't change *interp−>result* or clear error state. **Tcl_FreeResult** is most commonly used when a procedure is about to replace one result value with another.

**DIRECT ACCESS TO INTERP−>RESULT IS DEPRECATED**

It used to be legal for programs to directly read and write *interp−>result* to manipulate the interpreter result. Direct access to *interp−>result* is now strongly deprecated because it can make the result's string and object forms inconsistent. Programs should always read the result using the procedures **Tcl_GetObjResult** or **Tcl_GetStringResult**, and write the result using **Tcl_SetObjResult** or **Tcl_SetResult**.

**THE TCL_FREEPROC ARGUMENT TO TCL_SETRESULT**

**Tcl_SetResult**'s *freeProc* argument specifies how the Tcl system is to manage the storage for the *string* argument. If **Tcl_SetResult** or **Tcl_SetObjResult** are called at a time when *interp* holds a string result, they do whatever is necessary to dispose of the old string result (see the **Tcl_Interp** manual entry for details on this).

If *freeProc* is **TCL_STATIC** it means that *string* refers to an area of static storage that is guaranteed not to be modified until at least the next call to **Tcl_Eval**. If *freeProc* is **TCL_DYNAMIC** it means that *string* was allocated with a call to **Tcl_Alloc** and is now the property of the Tcl system. **Tcl_SetResult** will arrange for the string's storage to be released by calling **Tcl_Free** when it is no longer needed. If *freeProc* is **TCL_VOLATILE** it means that *string* points to an area of memory that is likely to be overwritten when **Tcl_SetResult** returns (e.g. it points to something in a stack frame). In this case **Tcl_SetResult** will make a copy of the string in dynamically allocated storage and arrange for the copy to be the result for the current Tcl command.

If *freeProc* isn't one of the values **TCL_STATIC**, **TCL_DYNAMIC**, and **TCL_VOLATILE**, then it is the address of a procedure that Tcl should call to free the string. This allows applications to use non−standard storage allocators. When Tcl no longer needs the storage for the string, it will call *freeProc*. *FreeProc* should have arguments and result that match the type **Tcl_FreeProc**:

```c
typedef void Tcl_FreeProc(char *blockPtr);
```

When *freeProc* is called, its *blockPtr* will be set to the value of *string* passed to **Tcl_SetResult**.
SEE ALSO

Tcl_AddErrorInfo, Tcl_CreateObjCommand, Tcl_SetErrorCode, Tcl_Interp

KEYWORDS

append, command, element, list, object, result, return value, interpreter

Copyright © 1989-1993 The Regents of the University of California.
Copyright © 1994-1997 Sun Microsystems, Inc.
Copyright © 1995-1997 Roger E. Critchlow Jr.
SetVar

NAME

Tcl_SetVar2Ex, Tcl_SetVar, Tcl_SetVar2, Tcl_ObjSetVar2, Tcl_GetVar2Ex, Tcl_GetVar, Tcl_GetVar2, Tcl_ObjGetVar2, Tcl_UnsetVar, Tcl_UnsetVar2 – manipulate Tcl variables

SYNOPSIS

ARGUMENTS

DESCRIPTION

TCL_GLOBAL_ONLY
TCL_NAMESPACE_ONLY
TCL_LEAVE_ERR_MSG
TCL_APPEND_VALUE
TCL_LIST_ELEMENT

SEE ALSO

KEYWORDS

NAME

Tcl_SetVar2Ex, Tcl_SetVar, Tcl_SetVar2, Tcl_ObjSetVar2, Tcl_GetVar2Ex, Tcl_GetVar, Tcl_GetVar2, Tcl_ObjGetVar2, Tcl_UnsetVar, Tcl_UnsetVar2 – manipulate Tcl variables

SYNOPSIS

#include <tcl.h>
Tcl_Obj *
Tcl_SetVar2Ex(interp, name1, name2, newValuePtr, flags)
CONST char *
Tcl_SetVar(interp, varName, newValue, flags)
CONST char *
Tcl_SetVar2(interp, name1, name2, newValue, flags)
Tcl_Obj *
Tcl_ObjSetVar2(interp, part1Ptr, part2Ptr, newValuePtr, flags)
Tcl_Obj *
Tcl_GetVar2Ex(interp, name1, name2, flags)
CONST char *
Tcl_GetVar(interp, varName, flags)
CONST char *
Tcl_GetVar2(interp, name1, name2, flags)
Tcl_Obj *
Tcl_ObjGetVar2(interp, part1Ptr, part2Ptr, flags)
int
Tcl_UnsetVar(interp, varName, flags)
int
Tcl_UnsetVar2(interp, name1, name2, flags)

ARGUMENTS

_Tcl_Interp *interp (in)
  Interpreter containing variable.
_CONST char *name1 (in)
  Contains the name of an array variable (if name2 is non-NULL) or (if name2 is NULL) either the
  name of a scalar variable or a complete name including both variable name and index. May include ::
  namespace qualifiers to specify a variable in a particular namespace.
_CONST char *name2 (in)
  If non-NULL, gives name of element within array; in this case name1 must refer to an array variable.
_Tcl_Obj *newValuePtr (in)
  Points to a Tcl object containing the new value for the variable.
_int flags (in)
  OR-ed combination of bits providing additional information. See below for valid values.
_CONST char *varName (in)
  Name of variable. May include :: namespace qualifiers to specify a variable in a particular
  namespace. May refer to a scalar variable or an element of an array.
_CONST char *newValue (in)
  New value for variable, specified as a null-terminated string. A copy of this value is stored in the
  variable.
_Tcl_Obj *part1Ptr (in)
  Points to a Tcl object containing the variable's name. The name may include a series of :: namespace
  qualifiers to specify a variable in a particular namespace. May refer to a scalar variable or an element
  of an array variable.
_Tcl_Obj *part2Ptr (in)
  If non-NULL, points to an object containing the name of an element within an array and part1Ptr
  must refer to an array variable.

DESCRIPTION

These procedures are used to create, modify, read, and delete Tcl variables from C code.

_Tcl_SetVar2Ex, Tcl_SetVar, Tcl_SetVar2, and Tcl_ObjSetVar2 will create a new variable or modify an
existing one. These procedures set the given variable to the value given by newValuePtr or newValue and
return a pointer to the variable's new value, which is stored in Tcl's variable structure. Tcl_SetVar2Ex and
_Tcl_ObjSetVar2 take the new value as a Tcl_Obj and return a pointer to a Tcl_Obj. Tcl_SetVar and
_Tcl_SetVar2 take the new value as a string and return a string; they are usually less efficient than
_Tcl_ObjSetVar2. Note that the return value may be different than the newValuePtr or newValue argument,
due to modifications made by write traces. If an error occurs in setting the variable (e.g. an array variable is
referenced without giving an index into the array) NULL is returned and an error message is left in interp's
result if the TCL_LEAVE_ERR_MSG flag bit is set.
Tcl_GetVar2Ex, Tcl_GetVar, Tcl_GetVar2, and Tcl_ObjGetVar2 return the current value of a variable. The arguments to these procedures are treated in the same way as the arguments to the procedures described above. Under normal circumstances, the return value is a pointer to the variable's value. For Tcl_GetVar2Ex and Tcl_ObjGetVar2 the value is returned as a pointer to a Tcl_Obj. For Tcl_GetVar and Tcl_GetVar2 the value is returned as a string; this is usually less efficient, so Tcl_GetVar2Ex or Tcl_ObjGetVar2 are preferred. If an error occurs while reading the variable (e.g. the variable doesn't exist or an array element is specified for a scalar variable), then NULL is returned and an error message is left in interp's result if the TCL_LEAVE_ERR_MSG flag bit is set.

Tcl_UnsetVar and Tcl_UnsetVar2 may be used to remove a variable, so that future attempts to read the variable will return an error. The arguments to these procedures are treated in the same way as the arguments to the procedures above. If the variable is successfully removed then TCL_OK is returned. If the variable cannot be removed because it doesn't exist then TCL_ERROR is returned and an error message is left in interp's result if the TCL_LEAVE_ERR_MSG flag bit is set. If an array element is specified, the given element is removed but the array remains. If an array name is specified without an index, then the entire array is removed.

The name of a variable may be specified to these procedures in four ways:

[1] If Tcl_SetVar, Tcl_GetVar, or Tcl_UnsetVar is invoked, the variable name is given as a single string, varName. If varName contains an open parenthesis and ends with a close parenthesis, then the value between the parentheses is treated as an index (which can have any string value) and the characters before the first open parenthesis are treated as the name of an array variable. If varName doesn't have parentheses as described above, then the entire string is treated as the name of a scalar variable.

[2] If the name1 and name2 arguments are provided and name2 is non-NULL, then an array element is specified and the array name and index have already been separated by the caller: name1 contains the name and name2 contains the index. An error is generated if name1 contains an open parenthesis and ends with a close parenthesis (array element) and name2 is non-NULL.

[3] If name2 is NULL, name1 is treated just like varName in case [1] above (it can be either a scalar or an array element variable name).

The flags argument may be used to specify any of several options to the procedures. It consists of an OR-ed combination of the following bits.

TCL_GLOBAL_ONLY
Under normal circumstances the procedures look up variables as follows. If a procedure call is active in interp, the variable is looked up at the current level of procedure call. Otherwise, the variable is looked up first in the current namespace, then in the global namespace. However, if this bit is set in flags, then the variable is looked up only in the global namespace even if there is a procedure call active. If both TCL_GLOBAL_ONLY and TCL_NAMESPACE_ONLY are given, TCL_GLOBAL_ONLY is ignored.
**TCLNAMESPACEONLY**
If this bit is set in *flags* then the variable is looked up only in the current namespace; if a procedure is active its variables are ignored, and the global namespace is also ignored unless it is the current namespace.

**TCL_LEAVE_ERR_MSG**
If an error is returned and this bit is set in *flags*, then an error message will be left in the interpreter's result, where it can be retrieved with `Tcl_GetObjResult` or `Tcl_GetStringResult`. If this flag bit isn't set then no error message is left and the interpreter's result will not be modified.

**TCL_APPEND_VALUE**
If this bit is set then *newValuePtr* or *newValue* is appended to the current value instead of replacing it. If the variable is currently undefined, then the bit is ignored. This bit is only used by the `Tcl_Set*` procedures.

**TCL_LIST_ELEMENT**
If this bit is set, then *newValue* is converted to a valid Tk list element before setting (or appending to) the variable. A separator space is appended before the new list element unless the list element is going to be the first element in a list or sublist (i.e. the variable's current value is empty, or contains the single character `"{}`, or ends in `"}`). When appending, the original value of the variable must also be a valid list, so that the operation is the appending of a new list element onto a list.

`Tcl_GetVar` and `Tcl_GetVar2` return the current value of a variable. The arguments to these procedures are treated in the same way as the arguments to `Tcl_SetVar` and `Tcl_SetVar2`. Under normal circumstances, the return value is a pointer to the variable's value (which is stored in Tcl's variable structure and will not change before the next call to `Tcl_SetVar` or `Tcl_SetVar2`). `Tcl_GetVar` and `Tcl_GetVar2` use the flag bits `TCL_GLOBALONLY` and `TCL_LEAVE_ERR_MSG`, both of which have the same meaning as for `Tcl_SetVar`. If an error occurs in reading the variable (e.g. the variable doesn't exist or an array element is specified for a scalar variable), then NULL is returned.

`Tcl_UnsetVar` and `Tcl_UnsetVar2` may be used to remove a variable, so that future calls to `Tcl_GetVar` or `Tcl_GetVar2` for the variable will return an error. The arguments to these procedures are treated in the same way as the arguments to `Tcl_GetVar` and `Tcl_GetVar2`. If the variable is successfully removed then TCL_OK is returned. If the variable cannot be removed because it doesn't exist then TCL_ERROR is returned. If an array element is specified, the given element is removed but the array remains. If an array name is specified without an index, then the entire array is removed.

**SEE ALSO**

`Tcl_GetObjResult`, `Tcl_GetStringResult`, `Tcl_TraceVar`

**KEYWORDS**

array, get variable, interpreter, object, scalar, set, unset, variable

Copyright © 1989–1993 The Regents of the University of California.  
Copyright © 1994–1997 Sun Microsystems, Inc.  
Signal

NAME

Tcl_SignalId, Tcl_SignalMsg – Convert signal codes

SYNOPSIS

#include <tcl.h>
CONST char *
Tcl_SignalId(sig)
CONST char *
Tcl_SignalMsg(sig)

ARGUMENTS

int sig (in)
A POSIX signal number such as SIGPIPE.

DESCRIPTION

Tcl_SignalId and Tcl_SignalMsg return a string representation of the provided signal number (sig).
Tcl_SignalId returns a machine-readable textual identifier such as "SIGPIPE". Tcl_SignalMsg returns a human-readable string such as "bus error". The strings returned by these functions are statically allocated and the caller must not free or modify them.

KEYWORDS

signals, signal numbers

Copyright ' 2001 ActiveState Tool Corp.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
Sleep

NAME

Tcl_Sleep – delay execution for a given number of milliseconds

SYNOPSIS

#include <tcl.h>
Tcl_Sleep(ms)

ARGUMENTS

int ms (in)
    Number of milliseconds to sleep.

DESCRIPTION

This procedure delays the calling process by the number of milliseconds given by the ms parameter and returns after that time has elapsed. It is typically used for things like flashing a button, where the delay is short and the application needn't do anything while it waits. For longer delays where the application needs to respond to other events during the delay, the procedure Tcl_CreateTimerHandler should be used instead of Tcl_Sleep.

KEYWORDS

sleep, time, wait

Copyright ' 1990 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
SourceRCFile

NAME

Tcl_SourceRCFile – source the Tcl rc file

SYNOPSIS

#include <tcl.h>
void
Tcl_SourceRCFile(interp)

ARGUMENTS

Tcl_Interp *interp (in)
    Tcl interpreter to source rc file into.

DESCRIPTION

Tcl_SourceRCFile is used to source the Tcl rc file at startup. It is typically invoked by Tcl_Main or Tk_Main. The name of the file sourced is obtained from the global variable tcl_rcFileName in the interpreter given by interp. If this variable is not defined, or if the file it indicates cannot be found, no action is taken.

On the Macintosh, after sourcing the rc file, this function will additionally source the TEXT resource indicated by the global variable tcl_rcRsrcName in interp.

KEYWORDS

application-specific initialization, main program, rc file

Copyright ' 1995-1997 Roger E. Critchlow Jr.
SplitList

NAME

Tcl_SplitList, Tcl_Merge, Tcl_ScanElement, Tcl_ConvertElement, Tcl_ScanCountedElement, Tcl_ConvertCountedElement – manipulate Tcl lists

SYNOPSIS

#include <tcl.h>

int
Tcl_SplitList(interp, list, argcPtr, argvPtr)
char *
Tcl_Merge(argc, argv)
int
Tcl_ScanElement(src, flagsPtr)
int
Tcl_ScanCountedElement(src, length, flagsPtr)
int
Tcl_ConvertElement(src, dst, flags)
int
Tcl_ConvertCountedElement(src, length, dst, flags)

ARGUMENTS

Tcl_Interp *interp (out)
Interpreter to use for error reporting. If NULL, then no error message is left.
char *list (in)
Pointer to a string with proper list structure.
int *argcPtr (out)
Filled in with number of elements in list.
CONST char ***argvPtr (out)
*argvPtr will be filled in with the address of an array of pointers to the strings that are the extracted elements of list. There will be *argePtr valid entries in the array, followed by a NULL entry.
int arge (in)
Number of elements in argv.
CONST char * CONST *argv (in)
Array of strings to merge together into a single list. Each string will become a separate element of the list.
CONST char *src (in)
String that is to become an element of a list.
int *flagsPtr (in)
Pointer to word to fill in with information about src. The value of *flagsPtr must be passed to Tcl_ConvertElement.
int \textit{length} (in)
\hspace{1em} \text{Number of bytes in string } \textit{src}.

\textit{char \*dst} (in)
\hspace{1em} \text{Place to copy converted list element. Must contain enough characters to hold converted string.}

\textit{int flags} (in)
\hspace{1em} \text{Information about } \textit{src}. \text{Must be value returned by previous call to } \text{Tcl\_ScanElement}, \text{possibly OR–ed with } \text{TCL\_DONT\_USE\_BRACES}.

\section*{DESCRIPTION}

These procedures may be used to disassemble and reassemble Tcl lists. \text{Tcl\_SplitList} breaks a list up into its constituent elements, returning an array of pointers to the elements using \textit{argcPtr} and \textit{argvPtr}. While extracting the arguments, \text{Tcl\_SplitList} obeys the usual rules for backslash substitutions and braces. The area of memory pointed to by \textit{argvPtr} is dynamically allocated; in addition to the array of pointers, it also holds copies of all the list elements. It is the caller's responsibility to free up all of this storage. For example, suppose that you have called \text{Tcl\_SplitList} with the following code:

\begin{verbatim}
int argc, code;
char *string;
char **argv;
...
code = Tcl_SplitList(interp, string, &argc, &argv);
\end{verbatim}

Then you should eventually free the storage with a call like the following:

\begin{verbatim}
Tcl_Free((char *) argv);
\end{verbatim}

\text{Tcl\_SplitList} normally returns \text{TCL\_OK}, which means the list was successfully parsed. If there was a syntax error in \textit{list}, then \text{TCL\_ERROR} is returned and the interpreter's result will point to an error message describing the problem (if \textit{interp} was not NULL). If \text{TCL\_ERROR} is returned then no memory is allocated and \textit{argvPtr} is not modified.

\text{Tcl\_Merge} is the inverse of \text{Tcl\_SplitList}: it takes a collection of strings given by \textit{argc} and \textit{argv} and generates a result string that has proper list structure. This means that commands like \text{index} may be used to extract the original elements again. In addition, if the result of \text{Tcl\_Merge} is passed to \text{Tcl\_Eval}, it will be parsed into \textit{argc} words whose values will be the same as the \textit{argv} strings passed to \text{Tcl\_Merge}. \text{Tcl\_Merge} will modify the list elements with braces and/or backslashes in order to produce proper Tcl list structure. The result string is dynamically allocated using \text{Tcl\_Alloc}; the caller must eventually release the space using \text{Tcl\_Free}.

If the result of \text{Tcl\_Merge} is passed to \text{Tcl\_SplitList}, the elements returned by \text{Tcl\_SplitList} will be identical to those passed into \text{Tcl\_Merge}. However, the converse is not true: if \text{Tcl\_SplitList} is passed a given string, and the resulting \textit{argc} and \textit{argv} are passed to \text{Tcl\_Merge}, the resulting string may not be the same as the original string passed to \text{Tcl\_SplitList}. This is because \text{Tcl\_Merge} may use backslashes and braces differently than the original string.
Tcl\_ScanElement and Tcl\_ConvertElement are the procedures that do all of the real work of Tcl\_Merge. Tcl\_ScanElement scans its src argument and determines how to use backslashes and braces when converting it to a list element. It returns an overestimate of the number of characters required to represent src as a list element, and it stores information in *flagsPtr that is needed by Tcl\_ConvertElement.

Tcl\_ConvertElement is a companion procedure to Tcl\_ScanElement. It does the actual work of converting a string to a list element. Its flags argument must be the same as the value returned by Tcl\_ScanElement. Tcl\_ConvertElement writes a proper list element to memory starting at *dst and returns a count of the total number of characters written, which will be no more than the result returned by Tcl\_ScanElement. Tcl\_ConvertElement writes out only the actual list element without any leading or trailing spaces: it is up to the caller to include spaces between adjacent list elements.

Tcl\_ConvertElement uses one of two different approaches to handle the special characters in src. Wherever possible, it handles special characters by surrounding the string with braces. This produces clean−looking output, but can't be used in some situations, such as when src contains unmatched braces. In these situations, Tcl\_ConvertElement handles special characters by generating backslash sequences for them. The caller may insist on the second approach by OR−ing the flag value returned by Tcl\_ScanElement with TCL\_DONT\_USE\_BRACES. Although this will produce an uglier result, it is useful in some special situations, such as when Tcl\_ConvertElement is being used to generate a portion of an argument for a Tcl command. In this case, surrounding src with curly braces would cause the command not to be parsed correctly.

Tcl\_ScanCountedElement and Tcl\_ConvertCountedElement are the same as Tcl\_ScanElement and Tcl\_ConvertElement, except the length of string src is specified by the length argument, and the string may contain embedded nulls.

KEYWORDS

backslash, convert, element, list, merge, split, strings

Copyright ' 1989−1993 The Regents of the University of California.
Copyright ' 1994−1996 Sun Microsystems, Inc.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
SplitPath

NAME

Tcl_SplitPath, Tcl_JoinPath, Tcl_GetPathType – manipulate platform-dependent file paths

SYNOPSIS

```
#include <tcl.h>
Tcl_SplitPath(path, argcPtr, argvPtr)
char *
Tcl_JoinPath(argc, argv, resultPtr)
Tcl_PathType
Tcl_GetPathType(path)
```

ARGUMENTS

```
CONST char * CONST *argvPtr (in)
    File path in a form appropriate for the current platform (see the filename manual entry for acceptable forms for path names).

int *argcPtr (out)
    Filled in with number of path elements in path.

CONST char ***argvPtr (out)
    *argvPtr will be filled in with the address of an array of pointers to the strings that are the extracted elements of path. There will be *argcPtr valid entries in the array, followed by a NULL entry.

int argc (in)
    Number of elements in argv.

CONST char * CONST *argv (in)
    Array of path elements to merge together into a single path.

Tcl_DString *resultPtr (in/out)
    A pointer to an initialized Tcl_DString to which the result of Tcl_JoinPath will be appended.
```

DESCRIPTION

These procedures have been superceded by the objectified procedures in the FileSystem man page, which are more efficient.

These procedures may be used to disassemble and reassemble file paths in a platform independent manner: they provide C−level access to the same functionality as the file split, file join, and file pathtype commands.

Tcl_SplitPath breaks a path into its constituent elements, returning an array of pointers to the elements using argcPtr and argvPtr. The area of memory pointed to by argvPtr is dynamically allocated; in addition to the array of pointers, it also holds copies of all the path elements. It is the caller's responsibility to free all of this storage. For example, suppose that you have called Tcl_SplitPath with the following code:
int argc;
char *path;
char **argv;
...
Tcl_SplitPath(string, &argc, &argv);

Then you should eventually free the storage with a call like the following:

Tcl_Free((char *) argv);

Tcl_JoinPath is the inverse of Tcl_SplitPath: it takes a collection of path elements given by argc and argv and generates a result string that is a properly constructed path. The result string is appended to resultPtr. ResultPtr must refer to an initialized Tcl_DString.

If the result of Tcl_SplitPath is passed to Tcl_JoinPath, the result will refer to the same location, but may not be in the same form. This is because Tcl_SplitPath and Tcl_JoinPath eliminate duplicate path separators and return a normalized form for each platform.

Tcl_GetPathType returns the type of the specified path, where Tcl_PathType is one of TCL_PATH_ABSOLUTE, TCL_PATH_RELATIVE, or TCL_PATH_VOLUME_RELATIVE. See the filename manual entry for a description of the path types for each platform.

KEYWORDS

file, filename, join, path, split, type

Copyright ' 1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
ChnlStack

NAME

Tcl_StackChannel, Tcl_UnstackChannel, Tcl_GetStackedChannel, Tcl_GetTopChannel – stack an I/O channel on top of another, and undo it

SYNOPSIS

#include <tcl.h>
Tcl_Channel
Tcl_StackChannel(interp, typePtr, clientData, mask, channel)
int
Tcl_UnstackChannel(interp, channel)
Tcl_Channel
Tcl_GetStackedChannel(channel)
Tcl_Channel
Tcl_GetTopChannel(channel)

ARGUMENTS

Tcl_Interp *interp (in)
Interpreter for error reporting.
Tcl_ChannelType *typePtr (in)
The new channel I/O procedures to use for channel.
ClientData clientData (in)
Arbitrary one-word value to pass to channel I/O procedures.
int mask (in)
Conditions under which channel will be used: OR-ed combination of TCL_READABLE, TCL_WRITABLE and TCL_EXCEPTION. This can be a subset of the operations currently allowed on channel.
Tcl_Channel channel (in)
An existing Tcl channel such as returned by Tcl_CreateChannel.

DESCRIPTION

These functions are for use by extensions that add processing layers to Tcl I/O channels. Examples include compression and encryption modules. These functions transparently stack and unstack a new channel on top of an existing one. Any number of channels can be stacked together.

The implementation of the Tcl channel code was rewritten in 8.3.2 to correct some problems with the previous implementation with regard to stacked channels. Anyone using stacked channels or creating stacked channel drivers should update to the new TCL_CHANNEL_VERSION_2 Tcl_ChannelType structure. See Tcl_CreateChannel for details.
**TclStackChannel** stacks a new *channel* on an existing channel with the same name that was registered for *channel* by **Tcl_RegisterChannel**.

**TclStackChannel** works by creating a new channel structure and placing itself on top of the channel stack. EOL translation, encoding and buffering options are shared between all channels in the stack. The hidden channel does no buffering, newline translations, or character set encoding. Instead, the buffering, newline translations, and encoding functions all remain at the top of the channel stack. A pointer to the new top channel structure is returned. If an error occurs when stacking the channel, NULL is returned instead.

The *mask* parameter specifies the operations that are allowed on the new channel. These can be a subset of the operations allowed on the original channel. For example, a read–write channel may become read–only after the **TclStackChannel** call.

Closing a channel closes the channels stacked below it. The close of stacked channels is executed in a way that allows buffered data to be properly flushed.

**TclUnstackChannel** reverses the process. The old channel is associated with the channel name, and the processing module added by **TclStackChannel** is destroyed. If there is no old channel, then **TclUnstackChannel** is equivalent to **TclClose**. If an error occurs unstacking the channel, **TCL_ERROR** is returned, otherwise **TCL_OK** is returned.

**TclGetTopChannel** returns the top channel in the stack of channels the supplied channel is part of.

**TclGetStackedChannel** returns the channel in the stack of channels which is just below the supplied channel.

**SEE ALSO**

Notifier, **TclCreateChannel**, **TclOpenFileChannel**, vwait(n).

**KEYWORDS**

channel, compression

---

Copyright ' 1999-2000 Ajuba Solutions.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
StaticPkg

NAME

Tcl_StaticPackage – make a statically linked package available via the load command

SYNOPSIS

#include <tcl.h>
Tcl_StaticPackage(interp, pkgName, initProc, safeInitProc)

ARGUMENTS

_Tcl_Interp *interp (in)
    If not NULL, points to an interpreter into which the package has already been loaded (i.e., the caller has already invoked the appropriate initialization procedure). NULL means the package hasn't yet been incorporated into any interpreter.

_CONST char *pkgName (in)
    Name of the package; should be properly capitalized (first letter upper-case, all others lower-case).

_Tcl_PackageInitProc *initProc (in)
    Procedure to invoke to incorporate this package into a trusted interpreter.

_Tcl_PackageInitProc *safeInitProc (in)
    Procedure to call to incorporate this package into a safe interpreter (one that will execute untrusted scripts). NULL means the package can't be used in safe interpreters.

DESCRIPTION

This procedure may be invoked to announce that a package has been linked statically with a Tcl application and, optionally, that it has already been loaded into an interpreter. Once Tcl_StaticPackage has been invoked for a package, it may be loaded into interpreters using the load command. Tcl_StaticPackage is normally invoked only by the Tcl_AppInit procedure for the application, not by packages for themselves (Tcl_StaticPackage should only be invoked for statically loaded packages, and code in the package itself should not need to know whether the package is dynamically or statically loaded).

When the load command is used later to load the package into an interpreter, one of initProc and safeInitProc will be invoked, depending on whether the target interpreter is safe or not. initProc and safeInitProc must both match the following prototype:

typedef int Tcl_PackageInitProc(_Tcl_Interp *interp);

The interp argument identifies the interpreter in which the package is to be loaded. The initialization procedure must return TCL_OK or TCL_ERROR to indicate whether or not it completed successfully; in the event of an error it should set the interpreter's result to point to an error message. The result or error from the initialization procedure will be returned as the result of the load command that caused the initialization
procedure to be invoked.

KEYWORDS

initialization procedure, package, static linking

Copyright © 1995-1996 Sun Microsystems, Inc.
Copyright © 1995-1997 Roger E. Critchlow Jr.
StrMatch

NAME

Tcl_StringMatch, Tcl_StringCaseMatch – test whether a string matches a pattern

SYNOPSIS

#include <tcl.h>

int
Tcl_StringMatch(string, pattern)

int
Tcl_StringCaseMatch(string, pattern, nocase)

ARGUMENTS

char *string (in)
    String to test.
char *pattern (in)
    Pattern to match against string. May contain special characters from the set *?\[].
int nocase (in)
    Specifies whether the match should be done case−sensitive (0) or case−insensitive (1).

DESCRIPTION

This utility procedure determines whether a string matches a given pattern. If it does, then Tcl_StringMatch returns 1. Otherwise Tcl_StringMatch returns 0. The algorithm used for matching is the same algorithm used in the ``string match'' Tcl command and is similar to the algorithm used by the C−shell for file name matching; see the Tcl manual entry for details.

In Tcl_StringCaseMatch, the algorithm is the same, but you have the option to make the matching case−insensitive. If you choose this (by passing nocase as 1), then the string and pattern are essentially matched in the lower case.

KEYWORDS

match, pattern, string

Copyright ' 1989-1993 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
StringObj

NAME

Tcl_NewStringObj, Tcl_NewUnicodeObj, Tcl_SetStringObj, Tcl_SetUnicodeObj, Tcl_GetStringFromObj,
Tcl_GetString, Tcl_GetUnicodeFromObj, Tcl_GetUnicode, Tcl_GetUniChar, Tcl_GetCharLength,
Tcl_GetRange, Tcl_AppendToObj, Tcl_AppendUnicodeToObj, Tcl_AppendStringsToObj,
Tcl_AppendStringsToObjVA, Tcl_AppendObjToObj, Tcl_SetObjLength, Tcl_ConcatObj,
Tcl_AttemptSetObjLength – manipulate Tcl objects as strings

SYNOPSIS

#include <tcl.h>
Tcl_Obj *
Tcl_NewStringObj(bytes, length)
Tcl_Obj *
Tcl_NewUnicodeObj(unicode, numChars)
void
Tcl_SetStringObj(objPtr, bytes, length)
void
Tcl_SetUnicodeObj(objPtr, unicode, numChars)
char *
Tcl_GetStringFromObj(objPtr, lengthPtr)
char *
Tcl_GetString(objPtr)
Tcl_UniChar *
Tcl_GetUnicodeFromObj(objPtr, lengthPtr)
Tcl_UniChar *
Tcl_GetUnicode(objPtr)
Tcl_UniChar
Tcl_GetUniChar(objPtr, index)
int
Tcl_GetCharLength(objPtr)
Tcl_Obj *
Tcl_GetRange(objPtr, first, last)
void
Tcl_AppendToObj(objPtr, bytes, length)
void
Tcl_AppendUnicodeToObj(objPtr, unicode, numChars)
void
Tcl_AppendObjToObj(objPtr, appendObjPtr)
void
Tcl_AppendStringsToObj(objPtr, string, string, ... (char *) NULL)
void
Tcl_AppendStringsToObjVA(objPtr, argList)
void
Tcl_SetObjLength(objPtr, newLength)
int
Tcl_AttemptSetObjLength(objPtr, newLength)
Tcl_Obj *
Tcl_ConcatObj(objc, objv)

ARGUMENTS

CONST char *bytes (in)
Points to the first byte of an array of UTF-8-encoded bytes used to set or append to a string object.
This byte array should not contain embedded null bytes unless length is negative. (Applications
needing null bytes should represent them as the two-byte sequence \700\600, use Tcl_ExternalToUtf
to convert, or Tcl_NewByteArrayListObj if the string is a collection of uninterpreted bytes.)

int length (in)
The number of bytes to copy from bytes when initializing, setting, or appending to a string object. If
negative, all bytes up to the first null are used.

CONST Tcl_UniChar *unicode (in)
Points to the first byte of an array of Unicode characters used to set or append to a string object. This
byte array may contain embedded null characters unless numChars is negative.

int numChars (in)
The number of Unicode characters to copy from unicode when initializing, setting, or appending to a
string object. If negative, all characters up to the first null character are used.

int index (in)
The index of the Unicode character to return.

int first (in)
The index of the first Unicode character in the Unicode range to be returned as a new object.

int last (in)
The index of the last Unicode character in the Unicode range to be returned as a new object.

Tcl_Obj *objPtr (in/out)
Points to an object to manipulate.

Tcl_Obj *appendObjPtr (in)
The object to append to objPtr in Tcl_AppendObjToObj.

int *lengthPtr (out)
If non-NULL, the location where Tcl_GetStringFromObj will store the the length of an object's
string representation.

CONST char *string (in)
Null-terminated string value to append to objPtr.

va_list argList (in)
An argument list which must have been initialised using TCL_VARARGS_START, and cleared
using va_end.

int newLength (in)
New length for the string value of objPtr, not including the final null character.

int objc (in)
The number of elements to concatenate.

\textit{Tcl\_Obj *objv[]} (in)

The array of objects to concatenate.

**DESCRIPTION**

The procedures described in this manual entry allow Tcl objects to be manipulated as string values. They use the internal representation of the object to store additional information to make the string manipulations more efficient. In particular, they make a series of append operations efficient by allocating extra storage space for the string so that it doesn't have to be copied for each append. Also, indexing and length computations are optimized because the Unicode string representation is calculated and cached as needed. When using the \textit{Tcl\_Append*} family of functions where the interpreter's result is the object being appended to, it is important to call \textit{Tcl\_Reset\_Result} first to ensure you are not unintentionally appending to existing data in the result object.

\textit{Tcl\_New\_String\_Obj} and \textit{Tcl\_Set\_String\_Obj} create a new object or modify an existing object to hold a copy of the string given by \textit{bytes} and \textit{length}. \textit{Tcl\_New\_Unicode\_Obj} and \textit{Tcl\_Set\_Unicode\_Obj} create a new object or modify an existing object to hold a copy of the Unicode string given by \textit{unicode} and \textit{numChars}.

\textit{Tcl\_New\_String\_Obj} and \textit{Tcl\_New\_Unicode\_Obj} return a pointer to a newly created object with reference count zero. All four procedures set the object to hold a copy of the specified string. \textit{Tcl\_Set\_String\_Obj} and \textit{Tcl\_Set\_Unicode\_Obj} free any old string representation as well as any old internal representation of the object.

\textit{Tcl\_Get\_String\_From\_Obj} and \textit{Tcl\_Get\_String} return an object's string representation. This is given by the returned byte pointer and (for \textit{Tcl\_Get\_String\_From\_Obj}) length, which is stored in \textit{lengthPtr} if it is non–NULL. If the object's UTF string representation is invalid (its byte pointer is NULL), the string representation is regenerated from the object's internal representation. The storage referenced by the returned byte pointer is owned by the object manager. It is passed back as a writable pointer so that extension author creating their own \textit{Tcl\_Obj\_Type} will be able to modify the string representation within the \textit{Tcl\_Update\_String\_Proc} of their \textit{Tcl\_Obj\_Type}. Except for that limited purpose, the pointer returned by \textit{Tcl\_Get\_String\_From\_Obj} or \textit{Tcl\_Get\_String} should be treated as read–only. It is recommended that this pointer be assigned to a (CONST char *) variable. Even in the limited situations where writing to this pointer is acceptable, one should take care to respect the copy–on–write semantics required by \textit{Tcl\_Obj}'s, with appropriate calls to \textit{Tcl\_Is\_Shared} and \textit{Tcl\_Duplicate\_Obj} prior to any in–place modification of the string representation. The procedure \textit{Tcl\_Get\_String} is used in the common case where the caller does not need the length of the string representation.

\textit{Tcl\_Get\_Unicode\_From\_Obj} and \textit{Tcl\_Get\_Unicode} return an object's value as a Unicode string. This is given by the returned pointer and (for \textit{Tcl\_Get\_Unicode\_From\_Obj}) length, which is stored in \textit{lengthPtr} if it is non–NULL. The storage referenced by the returned byte pointer is owned by the object manager and should not be modified by the caller. The procedure \textit{Tcl\_Get\_Unicode} is used in the common case where the caller does not need the length of the unicode string representation.

\textit{Tcl\_Get\_Uni\_Char} returns the \textit{index}'th character in the object's Unicode representation.

\textit{Tcl\_Get\_Range} returns a newly created object comprised of the characters between \textit{first} and \textit{last} (inclusive) in
the object's Unicode representation. If the object's Unicode representation is invalid, the Unicode
representation is regenerated from the object's string representation.

**Tcl_GetCharLength** returns the number of characters (as opposed to bytes) in the string object.

**Tcl_AppendToObj** appends the data given by *bytes* and *length* to the string representation of the object
specified by *objPtr*. If the object has an invalid string representation, then an attempt is made to convert *bytes*
is to the Unicode format. If the conversion is successful, then the converted form of *bytes* is appended to the
object's Unicode representation. Otherwise, the object's Unicode representation is invalidated and converted to
the UTF format, and *bytes* is appended to the object's new string representation.

**Tcl_AppendUnicodeToObj** appends the Unicode string given by *unicode* and *numChars* to the object
specified by *objPtr*. If the object has an invalid Unicode representation, then *unicode* is converted to the UTF
format and appended to the object's string representation. Appends are optimized to handle repeated appends
relatively efficiently (it overallocates the string or Unicode space to avoid repeated reallocations and copies of
object's string value).

**Tcl_AppendObjToObj** is similar to **Tcl_AppendToObj**, but it appends the string or Unicode value
(whichever exists and is best suited to be appended to *objPtr*) of *appendObjPtr* to *objPtr*.

**Tcl_AppendStringsToObj** is similar to **Tcl_AppendToObj** except that it can be passed more than one value
to append and each value must be a null-terminated string (i.e. none of the values may contain internal null
characters). Any number of *string* arguments may be provided, but the last argument must be a NULL pointer
to indicate the end of the list.

**Tcl_AppendStringsToObjVA** is the same as **Tcl_AppendStringsToObj** except that instead of taking a
variable number of arguments it takes an argument list.

The **Tcl_SetObjLength** procedure changes the length of the string value of its *objPtr* argument. If the
newLength argument is greater than the space allocated for the object's string, then the string space is
reallocated and the old value is copied to the new space; the bytes between the old length of the string and the
new length may have arbitrary values. If the newLength argument is less than the current length of the object's
string, with *objPtr*->*length* is reduced without reallocating the string space; the original allocated size for the
string is recorded in the object, so that the string length can be enlarged in a subsequent call to
**Tcl_SetObjLength** without reallocating storage. In all cases **Tcl_SetObjLength** leaves a null character at
*objPtr*->*bytes*[newLength].

**Tcl_AttemptSetObjLength** is identical in function to **Tcl_SetObjLength** except that if sufficient memory to
satisfy the request cannot be allocated, it does not cause the Tcl interpreter to panic. Thus, if newLength is
greater than the space allocated for the object's string, and there is not enough memory available to satisfy the
request, **Tcl_AttemptSetObjLength** will take no action and return 0 to indicate failure. If there is enough
memory to satisfy the request, **Tcl_AttemptSetObjLength** behaves just like **Tcl_SetObjLength** and returns
1 to indicate success.

The **Tcl_ConcatObj** function returns a new string object whose value is the space-separated concatenation of
the string representations of all of the objects in the `objv` array. `Tcl_ConcatObj` eliminates leading and trailing white space as it copies the string representations of the `objv` array to the result. If an element of the `objv` array consists of nothing but white space, then that object is ignored entirely. This white–space removal was added to make the output of the `concat` command cleaner–looking. `Tcl_ConcatObj` returns a pointer to a newly–created object whose ref count is zero.

SEE ALSO

`Tcl_NewObj`, `Tcl_IncrRefCount`, `Tcl_DecrRefCount`

KEYWORDS

append, internal representation, object, object type, string object, string type, string representation, concat, concatenate, unicode

---

Copyright © 1994–1997 Sun Microsystems, Inc.
SubstObj

NAME

Tcl_SubstObj – perform substitutions on Tcl objects

SYNOPSIS

#include <tcl.h>
Tcl_Obj *Tcl_SubstObj(interp, objPtr, flags)

ARGUMENTS

Tcl_Interp *interp (in)
  Interpreter in which to execute Tcl scripts and lookup variables. If an error occurs, the interpreter's result is modified to hold an error message.
Tcl_Obj *objPtr (in)
  A Tcl object containing the string to perform substitutions on.
int flags (in)
  ORed combination of flag bits that specify which substitutions to perform. The flags TCL_SUBST_COMMANDS, TCL_SUBST_VARIABLES and TCL_SUBST_BACKSLASHES are currently supported, and TCL_SUBST_ALL is provided as a convenience for the common case where all substitutions are desired.

DESCRIPTION

The Tcl_SubstObj function is used to perform substitutions on strings in the fashion of the subst command. It gets the value of the string contained in objPtr and scans it, copying characters and performing the chosen substitutions as it goes to an output object which is returned as the result of the function. In the event of an error occurring during the execution of a command or variable substitution, the function returns NULL and an error message is left in interp's result.

Three kinds of substitutions are supported. When the TCL_SUBST_BACKSLASHES bit is set in flags, sequences that look like backslash substitutions for Tcl commands are replaced by their corresponding character.

When the TCL_SUBST_VARIABLES bit is set in flags, sequences that look like variable substitutions for Tcl commands are replaced by the contents of the named variable.

When the TCL_SUBST_COMMANDS bit is set in flags, sequences that look like command substitutions for Tcl commands are replaced by the result of evaluating that script. Where an uncaught continue exception occurs during the evaluation of a command substitution, an empty string is substituted for the command. Where an uncaught break exception occurs during the evaluation of a command substitution, the result of the...
whole substitution on objPtr will be truncated at the point immediately before the start of the command substitution, and no characters will be added to the result or substitutions performed after that point.

SEE ALSO

subst

KEYWORDS

backslash substitution, command substitution, variable substitution

Copyright ' 2001 Donal K. Fellows
TraceCmd

NAME

Tcl_CommandTraceInfo, Tcl_TraceCommand, Tcl_UntraceCommand – monitor renames and deletes of a command

SYNOPSIS

ARGUMENTS

DESCRIPTION

TCL_TRACE_RENAME
TCL_TRACE_DELETE

CALLING COMMANDS DURING TRACES

MULTIPLE TRACES

TCL_TRACE_DESTROYED_FLAG
TCL_INTERP_DESTROYED

BUGS

KEYWORDS

NAME

Tcl_CommandTraceInfo, Tcl_TraceCommand, Tcl_UntraceCommand – monitor renames and deletes of a command

SYNOPSIS

#include <tcl.h>

ClientData

Tcl_CommandTraceInfo(interp, cmdName, flags, proc, prevClientData)

int

Tcl_TraceCommand(interp, cmdName, flags, proc, clientData)

void

Tcl_UntraceCommand(interp, cmdName, flags, proc, clientData)

ARGUMENTS

Tcl_Interp *interp (in)

Interpreter containing the command.

CONST char *cmdName (in)

Name of command.

int flags (in)

OR–ed collection of the value TCL_TRACE_RENAME and TCL_TRACE_DELETE.

Tcl_CommandTraceProc *proc (in)

Procedure to call when specified operations occur to cmdName.
ClientData clientData (in)
Arbitrary argument to pass to proc.

ClientData prevClientData (in)
If non–NULL, gives last value returned by Tcl_CommandTraceInfo, so this call will return information about next trace. If NULL, this call will return information about first trace.

DESCRIPTION

Tcl_TraceCommand allows a C procedure to monitor operations performed on a Tcl command, so that the C procedure is invoked whenever the command is renamed or deleted. If the trace is created successfully then Tcl_TraceCommand returns TCL_OK. If an error occurred (e.g. cmdName specifies a non–existent command) then TCL_ERROR is returned and an error message is left in the interpreter's result.

The flags argument to Tcl_TraceCommand indicates when the trace procedure is to be invoked. It consists of an OR–ed combination of any of the following values:

- TCL_TRACE_RENAME
  Invoke proc whenever the command is renamed.
- TCL_TRACE_DELETE
  Invoke proc when the command is deleted.

Whenever one of the specified operations occurs to the command, proc will be invoked. It should have arguments and result that match the type Tcl_CommandTraceProc:

```c
typedef void Tcl_CommandTraceProc(
    ClientData clientData,
    Tcl_Interp *interp,
    CONST char *oldName,
    CONST char *newName,
    int flags);
```

The clientData and interp parameters will have the same values as those passed to Tcl_TraceCommand when the trace was created. ClientData typically points to an application–specific data structure that describes what to do when proc is invoked. OldName gives the name of the command being renamed, and newName gives the name that the command is being renamed to (or an empty string or NULL when the command is being deleted.) Flags is an OR–ed combination of bits potentially providing several pieces of information. One of the bits TCL_TRACE_RENAME and TCL_TRACE_DELETE will be set in flags to indicate which operation is being performed on the command. The bit TCL_TRACE_DESTROYED will be set in flags if the trace is about to be destroyed; this information may be useful to proc so that it can clean up its own internal data structures (see the section TCL_TRACE_DESTROYED below for more details). Lastly, the bit TCL_INTERP_DESTROYED will be set if the entire interpreter is being destroyed. When this bit is set, proc must be especially careful in the things it does (see the section TCL_INTERP_DESTROYED below).

Tcl_UntraceCommand may be used to remove a trace. If the command specified by interp, cmdName, and flags has a trace set with flags, proc, and clientData, then the corresponding trace is removed. If no such trace exists, then the call to Tcl_UntraceCommand has no effect. The same bits are valid for flags as for calls to...
Tcl_TraceCommand.

Tcl_CommandTraceInfo may be used to retrieve information about traces set on a given command. The return value from Tcl_CommandTraceInfo is the clientData associated with a particular trace. The trace must be on the command specified by the interp, cmdName, and flags arguments (note that currently the flags are ignored; flags should be set to 0 for future compatibility) and its trace procedure must the same as the proc argument. If the prevClientData argument is NULL then the return value corresponds to the first (most recently created) matching trace, or NULL if there are no matching traces. If the prevClientData argument isn't NULL, then it should be the return value from a previous call to Tcl_CommandTraceInfo. In this case, the new return value will correspond to the next matching trace after the one whose clientData matches prevClientData, or NULL if no trace matches prevClientData or if there are no more matching traces after it. This mechanism makes it possible to step through all of the traces for a given command that have the same proc.

CALLING COMMANDS DURING TRACES

During rename traces, the command being renamed is visible with both names simultaneously, and the command still exists during delete traces (if TCL_INTERP_DESTROYED is not set). However, there is no mechanism for signaling that an error occurred in a trace procedure, so great care should be taken that errors do not get silently lost.

MULTIPLE TRACES

It is possible for multiple traces to exist on the same command. When this happens, all of the trace procedures will be invoked on each access, in order from most–recently–created to least–recently–created. Attempts to delete the command during a delete trace will fail silently, since the command is already scheduled for deletion anyway. If the command being renamed is renamed by one of its rename traces, that renaming takes precedence over the one that triggered the trace and the collection of traces will not be reexecuted; if several traces rename the command, the last renaming takes precedence.

TCL_TRACE_DESTROYED FLAG

In a delete callback to proc, the TCL_TRACE_DESTROYED bit is set in flags.

TCL_INTERP_DESTROYED

When an interpreter is destroyed, unset traces are called for all of its commands. The TCL_INTERP_DESTROYED bit will be set in the flags argument passed to the trace procedures. Trace procedures must be extremely careful in what they do if the TCL_INTERP_DESTROYED bit is set. It is not safe for the procedures to invoke any Tcl procedures on the interpreter, since its state is partially deleted. All that trace procedures should do under these circumstances is to clean up and free their own internal data structures.
BUGS

Tcl doesn’t do any error checking to prevent trace procedures from misusing the interpreter during traces with TCL_INTERP_DESTROYED set.

KEYWORDS

clientData, trace, command

Copyright © 2002 Donal K. Fellows
Copyright © 1995-1997 Roger E. Critchlow Jr.
TraceVar

NAME

Tcl_TraceVar, Tcl_TraceVar2, Tcl_UntraceVar, Tcl_UntraceVar2, Tcl_VarTraceInfo, Tcl_VarTraceInfo2 – monitor accesses to a variable

SYNOPSIS

ARGUMENTS

DESCRIPTION

TCL_GLOBAL_ONLY
TCL_NAMESPACE_ONLY
TCL_TRACE_READS
TCL_TRACE_WRITES
TCL_TRACE_UNSETS
TCL_TRACE_ARRAY
TCL_TRACE_RESULT_DYNAMIC
TCL_TRACE_RESULT_OBJECT

TWO−PART NAMES
ACCESSING VARIABLES DURING TRACES
CALLBACK TIMING
WHOLE−ARRAY TRACES
MULTIPLE TRACES
ERROR RETURNS
RESTRICTIONS
UNDEFINED VARIABLES
TCL_TRACE_DESTROYED_FLAG
TCL_INTERP_DESTROYED
BUGS
KEYWORDS

NAME

Tcl_TraceVar, Tcl_TraceVar2, Tcl_UntraceVar, Tcl_UntraceVar2, Tcl_VarTraceInfo, Tcl_VarTraceInfo2 – monitor accesses to a variable

SYNOPSIS

#include <tcl.h>

int Tcl_TraceVar(interp, varName, flags, proc, clientData)

int Tcl_TraceVar2(interp, name1, name2, flags, proc, clientData)

Tcl_UntraceVar(interp, varName, flags, proc, clientData)
Tcl_UntraceVar2(interp, name1, name2, flags, proc, clientData)
ClientData
Tcl_VarTraceInfo(interp, varName, flags, proc, prevClientData)
ClientData
Tcl_VarTraceInfo2(interp, name1, name2, flags, proc, prevClientData)

ARGUMENTS

Tcl_Interp *interp (in)
Interpreter containing variable.

CONST char *varName (in)
Name of variable. May refer to a scalar variable, to an array variable with no index, or to an array variable with a parenthesized index.

int flags (in)
OR-ed combination of the values TCL_TRACE_READS, TCL_TRACE_WRITES, TCL_TRACE_UNSETS, TCL_TRACE_ARRAY, TCL_GLOBAL_ONLY, TCL_NAMESPACE_ONLY, TCL_TRACE_RESULT_DYNAMIC and TCL_TRACE_RESULT_OBJECT. Not all flags are used by all procedures. See below for more information.

Tcl_VarTraceProc *proc (in)
Procedure to invoke whenever one of the traced operations occurs.

ClientData clientData (in)
Arbitrary one-word value to pass to proc.

CONST char *name1 (in)
Name of scalar or array variable (without array index).

CONST char *name2 (in)
For a trace on an element of an array, gives the index of the element. For traces on scalar variables or on whole arrays, is NULL.

ClientData prevClientData (in)
If non-NULL, gives last value returned by Tcl_VarTraceInfo or Tcl_VarTraceInfo2, so this call will return information about next trace. If NULL, this call will return information about first trace.

DESCRIPTION

Tcl_TraceVar allows a C procedure to monitor and control access to a Tcl variable, so that the C procedure is invoked whenever the variable is read or written or unset. If the trace is created successfully then Tcl_TraceVar returns TCL_OK. If an error occurred (e.g. varName specifies an element of an array, but the actual variable isn't an array) then TCL_ERROR is returned and an error message is left in the interpreter's result.

The flags argument to Tcl_TraceVar indicates when the trace procedure is to be invoked and provides information for setting up the trace. It consists of an OR-ed combination of any of the following values:

TCL_GLOBAL_ONLY
Normally, the variable will be looked up at the current level of procedure call; if this bit is set then the variable will be looked up at global level, ignoring any active procedures.

**TCL_NAMESPACE_ONLY**

Normally, the variable will be looked up at the current level of procedure call; if this bit is set then the variable will be looked up in the current namespace, ignoring any active procedures.

**TCL_TRACE_READS**

Invoke proc whenever an attempt is made to read the variable.

**TCL_TRACE_WRITES**

Invoke proc whenever an attempt is made to modify the variable.

**TCL_TRACE_UNSETS**

Invoke proc whenever the variable is unset. A variable may be unset either explicitly by an `unset` command, or implicitly when a procedure returns (its local variables are automatically unset) or when the interpreter is deleted (all variables are automatically unset).

**TCL_TRACE_ARRAY**

Invoke proc whenever the array command is invoked. This gives the trace procedure a chance to update the array before array names or array get is called. Note that this is called before an array set, but that will trigger write traces.

**TCL_TRACE_RESULT_DYNAMIC**

The result of invoking the proc is a dynamically allocated string that will be released by the Tcl library via a call to `ckfree`. Must not be specified at the same time as TCL_TRACE_RESULT_OBJECT.

**TCL_TRACE_RESULT_OBJECT**

The result of invoking the proc is a Tcl_Obj* (cast to a char*) with a reference count of at least one. The ownership of that reference will be transferred to the Tcl core for release (when the core has finished with it) via a call to `Tcl_DecrRefCount`. Must not be specified at the same time as TCL_TRACE_RESULT_DYNAMIC.

Whenever one of the specified operations occurs on the variable, proc will be invoked. It should have arguments and result that match the type Tcl_VarTraceProc:

```c
typedef char *Tcl_VarTraceProc(
    ClientData, 
    Tcl_Interp *interp,
    name1*,
    name2*,
    flags);
```

The `clientData` and `interp` parameters will have the same values as those passed to `Tcl_TraceVar` when the trace was created. `ClientData` typically points to an application−specific data structure that describes what to do when proc is invoked. `Name1` and `name2` give the name of the traced variable in the normal two−part form (see the description of `Tcl_TraceVar2` below for details). `Flags` is an OR−ed combination of bits providing several pieces of information. One of the bits TCL_TRACE_READS, TCL_TRACE_WRITES, TCL_TRACE_ARRAY, or TCL_TRACE_UNSETS will be set in `flags` to indicate which operation is being performed on the variable. The bit TCL_GLOBAL_ONLY will be set whenever the variable being accessed is a global one not accessible from the current level of procedure call: the trace procedure will need to pass this flag back to variable−related procedures like `Tcl_GetVar` if it attempts to access the variable. The bit
TCL_NAMESPACE_ONLY will be set whenever the variable being accessed is a namespace one not accessible from the current level of procedure call: the trace procedure will need to pass this flag back to variable–related procedures like Tcl_GetVar if it attempts to access the variable. The bit TCL_TRACE_DESTROYED will be set in flags if the trace is about to be destroyed; this information may be useful to proc so that it can clean up its own internal data structures (see the section TCL_TRACE_DESTROYED below for more details). Lastly, the bit TCL_INTERP_DESTROYED will be set if the entire interpreter is being destroyed. When this bit is set, proc must be especially careful in the things it does (see the section TCL_INTERP_DESTROYED below). The trace procedure's return value should normally be NULL; see ERROR RETURNS below for information on other possibilities.

Tcl_UntraceVar may be used to remove a trace. If the variable specified by interp, varName, and flags has a trace set with flags, proc, and clientData, then the corresponding trace is removed. If no such trace exists, then the call to Tcl_UntraceVar has no effect. The same bits are valid for flags as for calls to Tcl_TraceVar.

Tcl_VarTraceInfo may be used to retrieve information about traces set on a given variable. The return value from Tcl_VarTraceInfo is the clientData associated with a particular trace. The trace must be on the variable specified by the interp, varName, and flags arguments (only the TCL_GLOBAL_ONLY and TCL_NAMESPACE_ONLY bits from flags is used; other bits are ignored) and its trace procedure must the same as the proc argument. If the prevClientData argument is NULL then the return value corresponds to the first (most recently created) matching trace, or NULL if there are no matching traces. If the prevClientData argument isn't NULL, then it should be the return value from a previous call to Tcl_VarTraceInfo. In this case, the new return value will correspond to the next matching trace after the one whose clientData matches prevClientData, or NULL if no trace matches prevClientData or if there are no more matching traces after it. This mechanism makes it possible to step through all of the traces for a given variable that have the same proc.

TWO–PART NAMES

The procedures Tcl_TraceVar2, Tcl_UntraceVar2, and Tcl_VarTraceInfo2 are identical to Tcl_TraceVar, Tcl_UntraceVar, and Tcl_VarTraceInfo, respectively, except that the name of the variable consists of two parts. Name1 gives the name of a scalar variable or array, and name2 gives the name of an element within an array. When name2 is NULL, name1 may contain both an array and an element name: if the name contains an open parenthesis and ends with a close parenthesis, then the value between the parentheses is treated as an element name (which can have any string value) and the characters before the first open parenthesis are treated as the name of an array variable. If name2 is NULL and name1 does not refer to an array element it means that either the variable is a scalar or the trace is to be set on the entire array rather than an individual element (see WHOLE–ARRAY TRACES below for more information).

ACCESSING VARIABLES DURING TRACES

During read, write, and array traces, the trace procedure can read, write, or unset the traced variable using Tcl_GetVar2, Tcl_SetVar2, and other procedures. While proc is executing, traces are temporarily disabled for the variable, so that calls to Tcl_GetVar2 and Tcl_SetVar2 will not cause proc or other trace procedures to be invoked again. Disabling only occurs for the variable whose trace procedure is active; accesses to other variables will still be traced. However, if a variable is unset during a read or write trace then unset traces will
be invoked.

During unset traces the variable has already been completely expunged. It is possible for the trace procedure to read or write the variable, but this will be a new version of the variable. Traces are not disabled during unset traces as they are for read and write traces, but existing traces have been removed from the variable before any trace procedures are invoked. If new traces are set by unset trace procedures, these traces will be invoked on accesses to the variable by the trace procedures.

CALLBACK TIMING

When read tracing has been specified for a variable, the trace procedure will be invoked whenever the variable's value is read. This includes `set` Tcl commands, `$`–notation in Tcl commands, and invocations of the `Tcl_GetVar` and `Tcl_GetVar2` procedures. `Proc` is invoked just before the variable's value is returned. It may modify the value of the variable to affect what is returned by the traced access. If it unsets the variable then the access will return an error just as if the variable never existed.

When write tracing has been specified for a variable, the trace procedure will be invoked whenever the variable's value is modified. This includes `set` commands, commands that modify variables as side effects (such as `catch` and `scan`), and calls to the `Tcl_SetVar` and `Tcl_SetVar2` procedures). `Proc` will be invoked after the variable's value has been modified, but before the new value of the variable has been returned. It may modify the value of the variable to override the change and to determine the value actually returned by the traced access. If it deletes the variable then the traced access will return an empty string.

When array tracing has been specified, the trace procedure will be invoked at the beginning of the array command implementation, before any of the operations like get, set, or names have been invoked. The trace procedure can modify the array elements with `Tcl_SetVar` and `Tcl_SetVar2`.

When unset tracing has been specified, the trace procedure will be invoked whenever the variable is destroyed. The traces will be called after the variable has been completely unset.

WHOLE-ARRAY TRACES

If a call to `Tcl_TraceVar` or `Tcl_TraceVar2` specifies the name of an array variable without an index into the array, then the trace will be set on the array as a whole. This means that `proc` will be invoked whenever any element of the array is accessed in the ways specified by `flags`. When an array is unset, a whole-array trace will be invoked just once, with `name1` equal to the name of the array and `name2` NULL; it will not be invoked once for each element.

MULTIPLE TRACES

It is possible for multiple traces to exist on the same variable. When this happens, all of the trace procedures will be invoked on each access, in order from most–recently–created to least–recently–created. When there exist whole–array traces for an array as well as traces on individual elements, the whole–array traces are invoked before the individual–element traces. If a read or write trace unsets the variable then all of the unset traces will be invoked but the remainder of the read and write traces will be skipped.
ERROR RETURNS

Under normal conditions trace procedures should return NULL, indicating successful completion. If proc returns a non-NULL value it signifies that an error occurred. The return value must be a pointer to a static character string containing an error message, unless (exactly one of) the TCL_TRACE_RESULT_DYNAMIC and TCL_TRACE_RESULT_OBJECT flags is set, which specify that the result is either a dynamic string (to be released with ckfree) or a Tcl_Obj* (cast to char* and to be released with Tcl_DecrRefCount) containing the error message. If a trace procedure returns an error, no further traces are invoked for the access and the traced access aborts with the given message. Trace procedures can use this facility to make variables read-only, for example (but note that the value of the variable will already have been modified before the trace procedure is called, so the trace procedure will have to restore the correct value).

The return value from proc is only used during read and write tracing. During unset traces, the return value is ignored and all relevant trace procedures will always be invoked.

RESTRICTIONS

A trace procedure can be called at any time, even when there is a partially-formed result in the interpreter's result area. If the trace procedure does anything that could damage this result (such as calling Tcl_Eval) then it must save the original values of the interpreter's result and freeProc fields and restore them before it returns.

UNDEFINED VARIABLES

It is legal to set a trace on an undefined variable. The variable will still appear to be undefined until the first time its value is set. If an undefined variable is traced and then unset, the unset will fail with an error (``no such variable''), but the trace procedure will still be invoked.

TCL_TRACE_DESTROYED FLAG

In an unset callback to proc, the TCL_TRACE_DESTROYED bit is set in flags if the trace is being removed as part of the deletion. Traces on a variable are always removed whenever the variable is deleted; the only time TCL_TRACE_DESTROYED isn't set is for a whole-array trace invoked when only a single element of an array is unset.

TCL_INTERP_DESTROYED

When an interpreter is destroyed, unset traces are called for all of its variables. The TCL_INTERP_DESTROYED bit will be set in the flags argument passed to the trace procedures. Trace procedures must be extremely careful in what they do if the TCL_INTERP_DESTROYED bit is set. It is not safe for the procedures to invoke any Tcl procedures on the interpreter, since its state is partially deleted. All that trace procedures should do under these circumstances is to clean up and free their own internal data structures.
BUGS

Tcl doesn’t do any error checking to prevent trace procedures from misusing the interpreter during traces with TCL_INTERP_DESTROYED set.

Array traces are not yet integrated with the Tcl "info exists" command, nor is there Tcl–level access to array traces.

KEYWORDS

cientData, trace, variable

Copyright ' 1989–1993 The Regents of the University of California.
Copyright ' 1994–1996 Sun Microsystems, Inc.
Translate

NAME

Tcl_TranslateFileName – convert file name to native form and replace tilde with home directory

SYNOPSIS

#include <tcl.h>
char *
Tcl_TranslateFileName(interp, name, bufferPtr)

ARGUMENTS

_Tcl_Interp *interp (in)
  Interpreter in which to report an error, if any.
_CONST char *name (in)
  File name, which may start with a "~".
_Tcl_DString *bufferPtr (in/out)
  If needed, this dynamic string is used to store the new file name. At the time of the call it should be
  uninitialized or free. The caller must eventually call Tcl_DStringFree to free up anything stored here.

DESCRIPTION

This utility procedure translates a file name to a form suitable for passing to the local operating system. It
converts network names into native form and does tilde substitution.

If _Tcl_TranslateFileName has to do tilde substitution or translate the name then it uses the dynamic string at
*bufferPtr to hold the new string it generates. After _Tcl_TranslateFileName returns a non–NULL result, the
caller must eventually invoke Tcl_DStringFree to free any information placed in *bufferPtr. The caller need
not know whether or not _Tcl_TranslateFileName actually used the string; _Tcl_TranslateFileName
initializes *bufferPtr even if it doesn't use it, so the call to _Tcl_DStringFree will be safe in either case.

If an error occurs (e.g. because there was no user by the given name) then NULL is returned and an error
message will be left in the interpreter's result. When an error occurs, _Tcl_TranslateFileName frees the
dynamic string itself so that the caller need not call _Tcl_DStringFree.

The caller is responsible for making sure that the interpreter's result has its default empty value when
_Tcl_TranslateFileName is invoked.

SEE ALSO

filename
KEYWORDS

file name, home directory, tilde, translate, user

Copyright ' 1989-1993 The Regents of the University of California.
Copyright ' 1994-1998 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
UniCharIsAlpha

NAME

Tcl_UniCharIsAlnum, Tcl_UniCharIsAlpha, Tcl_UniCharIsControl, Tcl_UniCharIsDigit, Tcl_UniCharIsGraph, Tcl_UniCharIsLower, Tcl_UniCharIsPrint, Tcl_UniCharIsPunct, Tcl_UniCharIsSpace, Tcl_UniCharIsUpper, Tcl_UniCharIsWordChar – routines for classification of Tcl_UniChar characters

SYNOPSIS

#include <tcl.h>
int Tcl_UniCharIsAlnum(ch)
int Tcl_UniCharIsAlpha(ch)
int Tcl_UniCharIsControl(ch)
int Tcl_UniCharIsDigit(ch)
int Tcl_UniCharIsGraph(ch)
int Tcl_UniCharIsLower(ch)
int Tcl_UniCharIsPrint(ch)
int Tcl_UniCharIsPunct(ch)
int Tcl_UniCharIsSpace(ch)
int Tcl_UniCharIsUpper(ch)
int Tcl_UniCharIsWordChar(ch)

ARGUMENTS

int ch (in)
    The Tcl_UniChar to be examined.

DESCRIPTION

All of the routines described examine Tcl_UniChars and return a boolean value. A non–zero return value means that the character does belong to the character class associated with the called routine. The rest of this document just describes the character classes associated with the various routines.
Note: A \texttt{Tcl\_UniChar} is a Unicode character represented as an unsigned, fixed-size quantity.

**CHARACTER CLASSES**

\texttt{Tcl\_UniCharIsAlnum} tests if the character is an alphanumerical Unicode character.

\texttt{Tcl\_UniCharIsAlpha} tests if the character is an alphabetic Unicode character.

\texttt{Tcl\_UniCharIsControl} tests if the character is a Unicode control character.

\texttt{Tcl\_UniCharIsDigit} tests if the character is a numeric Unicode character.

\texttt{Tcl\_UniCharIsGraph} tests if the character is any Unicode print character except space.

\texttt{Tcl\_UniCharIsLower} tests if the character is a lowercase Unicode character.

\texttt{Tcl\_UniCharIsPrint} tests if the character is a Unicode print character.

\texttt{Tcl\_UniCharIsPunct} tests if the character is a Unicode punctuation character.

\texttt{Tcl\_UniCharIsSpace} tests if the character is a whitespace Unicode character.

\texttt{Tcl\_UniCharIsUpper} tests if the character is an uppercase Unicode character.

\texttt{Tcl\_UniCharIsWordChar} tests if the character is alphanumerical or a connector punctuation mark.

**KEYWORDS**

\texttt{unicode, classification}
UpVar

NAME

Tcl_UpVar, Tcl_UpVar2 – link one variable to another

SYNOPSIS

#include <tcl.h>
int Tcl_UpVar(interp, frameName, sourceName, destName, flags)
int Tcl_UpVar2(interp, frameName, name1, name2, destName, flags)

ARGUMENTS

Tcl_Interp *interp (in)
Interpreter containing variables; also used for error reporting.
CONST char *frameName (in)
Identifies the stack frame containing source variable. May have any of the forms accepted by the upvar command, such as #0 or 1.
CONST char *sourceName (in)
Name of source variable, in the frame given by frameName. May refer to a scalar variable or to an array variable with a parenthesized index.
CONST char *destName (in)
Name of destination variable, which is to be linked to source variable so that references to destName refer to the other variable. Must not currently exist except as an upvar-ed variable.
int flags (in)
Either TCL_GLOBAL_ONLY or 0; if non-zero, then destName is a global variable; otherwise it is a local to the current procedure (or global if no procedure is active).
CONST char *name1 (in)
First part of source variable's name (scalar name, or name of array without array index).
CONST char *name2 (in)
If source variable is an element of an array, gives the index of the element. For scalar source variables, is NULL.

DESCRIPTION

Tcl_UpVar and Tcl_UpVar2 provide the same functionality as the upvar command: they make a link from a source variable to a destination variable, so that references to the destination are passed transparently through to the source. The name of the source variable may be specified either as a single string such as xyx or a(24) (by calling Tcl_UpVar) or in two parts where the array name has been separated from the element name (by calling Tcl_UpVar2). The destination variable name is specified in a single string; it may not be an array element.
Both procedures return either TCL_OK or TCL_ERROR, and they leave an error message in the interpreter's result if an error occurs.

As with the \texttt{upvar} command, the source variable need not exist; if it does exist, unsetting it later does not destroy the link. The destination variable may exist at the time of the call, but if so it must exist as a linked variable.

\textbf{KEYWORDS}

\texttt{linked variable, upvar, variable}

\texttt{Copyright }'\texttt{ 1994 The Regents of the University of California.}
\texttt{Copyright }'\texttt{ 1994-1996 Sun Microsystems, Inc.}
\texttt{Copyright }'\texttt{ 1995-1997 Roger E. Critchlow Jr.}
ToUpper

NAME

Tcl_UniCharToUpper, Tcl_UniCharToLower, Tcl_UniCharToTitle, Tcl_UtfToUpper, Tcl_UtfToLower, Tcl_UtfToTitle – routines for manipulating the case of Unicode characters and UTF–8 strings.

SYNOPSIS

#include <tcl.h>
Tcl_UniChar
Tcl_UniCharToUpper(ch)
Tcl_UniChar
Tcl_UniCharToLower(ch)
Tcl_UniChar
Tcl_UniCharToTitle(ch)
int
Tcl_UtfToUpper(str)
int
Tcl_UtfToLower(str)
int
Tcl_UtfToTitle(str)

ARGUMENTS

int ch (in)

The Tcl_UniChar to be converted.

char *str (in/out)

Pointer to UTF–8 string to be converted in place.

DESCRIPTION

The first three routines convert the case of individual Unicode characters:

If ch represents a lower–case character, Tcl_UniCharToUpper returns the corresponding upper–case character. If no upper–case character is defined, it returns the character unchanged.

If ch represents an upper–case character, Tcl_UniCharToLower returns the corresponding lower–case character. If no lower–case character is defined, it returns the character unchanged.

If ch represents a lower–case character, Tcl_UniCharToTitle returns the corresponding title–case character. If no title–case character is defined, it returns the corresponding upper–case character. If no upper–case character is defined, it returns the character unchanged. Title–case is defined for a small number of characters that have a different appearance when they are at the beginning of a capitalized word.
The next three routines convert the case of UTF-8 strings in place in memory:

**Tcl_UtfToUpper** changes every UTF-8 character in *str* to upper-case. Because changing the case of a character may change its size, the byte offset of each character in the resulting string may differ from its original location. **Tcl_UtfToUpper** writes a null byte at the end of the converted string. **Tcl_UtfToUpper** returns the new length of the string in bytes. This new length is guaranteed to be no longer than the original string length.

**Tcl_UtfToLower** is the same as **Tcl_UtfToUpper** except it turns each character in the string into its lower-case equivalent.

**Tcl_UtfToTitle** is the same as **Tcl_UtfToUpper** except it turns the first character in the string into its title-case equivalent and all following characters into their lower-case equivalents.

**BUGS**

At this time, the case conversions are only defined for the ISO8859-1 characters. Unicode characters above 0x00ff are not modified by these routines.

**KEYWORDS**

utf, unicode, toupper, tolower, totitle, case
WrongNumArgs

NAME

Tcl_WrongNumArgs – generate standard error message for wrong number of arguments

SYNOPSIS

#include <tcl.h>
Tcl_WrongNumArgs(interp, objc, objv, message)

ARGUMENTS

Tcl_Interp interp (in)
   Interpreter in which error will be reported: error message gets stored in its result object.

int objc (in)
   Number of leading arguments from objv to include in error message.

Tcl_Obj *CONST objv[] (in)
   Arguments to command that had the wrong number of arguments.

CONST char *message (in)
   Additional error information to print after leading arguments from objv. This typically gives the acceptable syntax of the command. This argument may be NULL.

DESCRIPTION

Tcl_WrongNumArgs is a utility procedure that is invoked by command procedures when they discover that they have received the wrong number of arguments. Tcl_WrongNumArgs generates a standard error message and stores it in the result object of interp. The message includes the objc initial elements of objv plus message. For example, if objv consists of the values foo and bar, objc is 1, and message is `fileName count' then interp's result object will be set to the following string:

wrong # args: should be "foo fileName count"

If objc is 2, the result will be set to the following string:

wrong # args: should be "foo bar fileName count"

Objc is usually 1, but may be 2 or more for commands like string and the Tk widget commands, which use the first argument as a subcommand.

Some of the objects in the objv array may be abbreviations for a subcommand. The command Tcl_GetIndexFromObj will convert the abbreviated string object into an indexObject. If an error occurs in the parsing of the subcommand we would like to use the full subcommand name rather than the abbreviation. If the Tcl_WrongNumArgs command finds any indexObjects in the objv array it will use the full
subcommand name in the error message instead of the abbreviated name that was originally passed in. Using the above example, let's assume that *bar* is actually an abbreviation for *barfly* and the object is now an indexObject because it was passed to *Tcl_GetIndexFromObj*. In this case the error message would be:

```
wrong # args: should be "foo barfly fileName count"
```

**SEE ALSO**

*Tcl_GetIndexFromObj*

**KEYWORDS**

command, error message, wrong number of arguments
Thread

NAME

Tcl_ConditionNotify, Tcl_ConditionWait, Tcl_ConditionFinalize, Tcl_GetThreadData, Tcl_MutexLock, Tcl_MutexUnlock, Tcl_MutexFinalize, Tcl_CreateThread, Tcl_JoinThread – Tcl thread support.

SYNOPSIS

#include <tcl.h>

void
Tcl_ConditionNotify(condPtr)

void
Tcl_ConditionWait(condPtr, mutexPtr, timePtr)

void
Tcl_ConditionFinalize(condPtr)

Void *
Tcl_GetThreadData(keyPtr, size)

void
Tcl_MutexLock(mutexPtr)

void
Tcl_MutexUnlock(mutexPtr)

void
Tcl_MutexFinalize(mutexPtr)

int
Tcl_CreateThread(idPtr, threadProc, clientData, stackSize, flags)

int
Tcl_JoinThread(id, result)

ARGUMENTS

Tcl_Condition *condPtr (in)
   A condition variable, which must be associated with a mutex lock.

Tcl_Mutex *mutexPtr (in)
   A mutex lock.

Tcl_Time *timePtr (in)
   A time limit on the condition wait. NULL to wait forever. Note that a polling value of 0 seconds doesn't make much sense.

Tcl_ThreadDataKey *keyPtr (in)
   This identifies a block of thread local storage. The key should be static and process-wide, yet each thread will end up associating a different block of storage with this key.

int *size (in)
   The size of the thread local storage block. This amount of data is allocated and initialized to zero the first time each thread calls Tcl_GetThreadData.
Tcl_Threadld *idPtr (out)
The referred storage will contain the id of the newly created thread as returned by the operating
system.
Tcl_Threadld id (in)
Id of the thread waited upon.
Tcl_ThreadCreateProc threadProc (in)
This procedure will act as the main() of the newly created thread. The specified clientData will be its
sole argument.
ClientData clientData (in)
Arbitrary information. Passed as sole argument to the threadProc.
int stackSize (in)
The size of the stack given to the new thread.
int flags (in)
Bitmask containing flags allowing the caller to modify behaviour of the new thread.
int *result (out)
The referred storage is used to place the exit code of the thread waited upon into it.

INTRODUCTION

Beginning with the 8.1 release, the Tcl core is thread safe, which allows you to incorporate Tcl into
multithreaded applications without customizing the Tcl core. To enable Tcl multithreading support, you must
include the −−enable−threads option to configure when you configure and compile your Tcl core.

An important constraint of the Tcl threads implementation is that only the thread that created a Tcl interpreter
can use that interpreter. In other words, multiple threads can not access the same Tcl interpreter. (However,
as was the case in previous releases, a single thread can safely create and use multiple interpreters.)

Tcl does provide Tcl_CreateThread for creating threads. The caller can determine the size of the stack given
to the new thread and modify the behaviour through the supplied flags. The value
TCL_THREAD_STACK_DEFAULT for the stackSize indicates that the default size as specified by the
operating system is to be used for the new thread. As for the flags, currently are only the values
TCL_THREAD_NOFLAGS and TCL_THREAD_JOINABLE defined. The first of them invokes the
default behaviour with no specialties. Using the second value marks the new thread as joinable. This means
that another thread can wait for the such marked thread to exit and join it.

Restrictions: On some unix systems the pthread−library does not contain the functionality to specify the
stacksize of a thread. The specified value for the stacksize is ignored on these systems. Both Windows and
Macintosh currently do not support joinable threads. This flag value is therefore ignored on these platforms.

Tcl does provide Tcl.ExitThread and Tcl_FinalizeThread for terminating threads and invoking optional
per−thread exit handlers. See the Tcl_Exit page for more information on these procedures.

The Tcl.JoinThread function is provided to allow threads to wait upon the exit of another thread, which
must have been marked as joinable through usage of the TCL_THREAD_JOINABLE−flag during its
creation via Tcl_CreateThread.
Trying to wait for the exit of a non-joinable thread or a thread which is already waited upon will result in an error. Waiting for a joinable thread which already exited is possible, the system will retain the necessary information until after the call to `Tcl_JoinThread`. This means that not calling `Tcl_JoinThread` for a joinable thread will cause a memory leak.

Tcl provides `Tcl_ThreadQueueEvent` and `Tcl_ThreadAlert` for handling event queueing in multithreaded applications. See the `Notifier` manual page for more information on these procedures.

In this release, the Tcl language itself provides no support for creating multithreaded scripts (for example, scripts that could spawn a Tcl interpreter in a separate thread). If you need to add this feature at this time, see the `tclThreadTest.c` file in the Tcl source distribution for an experimental implementation or use the Tcl "Threading Extension" package implementing thread creation and management commands at the script level.

**DESCRIPTION**

A mutex is a lock that is used to serialize all threads through a piece of code by calling `Tcl_MutexLock` and `Tcl_MutexUnlock`. If one thread holds a mutex, any other thread calling `Tcl_MutexLock` will block until `Tcl_MutexUnlock` is called. A mutex can be destroyed after its use by calling `Tcl_MutexFinalize`. The result of locking a mutex twice from the same thread is undefined. On some platforms it will result in a deadlock.

The `Tcl_MutexLock`, `Tcl_MutexUnlock` and `Tcl_MutexFinalize` procedures are defined as empty macros if not compiling with threads enabled. For declaration of mutexes the `TCL_DECLARE_MUTEX` macro should be used. This macro assures correct mutex handling even when the core is compiled without threads enabled.

A condition variable is used as a signaling mechanism: a thread can lock a mutex and then wait on a condition variable with `Tcl_ConditionWait`. This atomically releases the mutex lock and blocks the waiting thread until another thread calls `Tcl_ConditionNotify`. The caller of `Tcl_ConditionNotify` should have the associated mutex held by previously calling `Tcl_MutexLock`, but this is not enforced. Notifying the condition variable unblocks all threads waiting on the condition variable, but they do not proceed until the mutex is released with `Tcl_MutexUnlock`. The implementation of `Tcl_ConditionWait` automatically locks the mutex before returning.

The caller of `Tcl_ConditionWait` should be prepared for spurious notifications by calling `Tcl_ConditionWait` within a while loop that tests some invariant.

A condition variable can be destroyed after its use by calling `Tcl_ConditionFinalize`.

The `Tcl_ConditionNotify`, `Tcl_ConditionWait` and `Tcl_ConditionFinalize` procedures are defined as empty macros if not compiling with threads enabled.

The `Tcl_GetThreadData` call returns a pointer to a block of thread–private data. Its argument is a key that is shared by all threads and a size for the block of storage. The storage is automatically allocated and initialized to all zeros the first time each thread asks for it. The storage is automatically deallocated by `Tcl_FinalizeThread`. 
INITIALIZATION

All of these synchronization objects are self initializing. They are implemented as opaque pointers that should be NULL upon first use. The mutexes and condition variables are either cleaned up by process exit handlers (if living that long) or explicitly by calls to Tcl_MutexFinalize or Tcl_ConditionFinalize. Thread local storage is reclaimed during Tcl_FinalizeThread.

CREATING THREADS

The API to create threads is not finalized at this time. There are private facilities to create threads that contain a new Tcl interpreter, and to send scripts among threads. Dive into tclThreadTest.c and tclThread.c for examples.

SEE ALSO

Tcl_GetCurrentThread, Tcl_ThreadQueueEvent, Tcl_ThreadAlert, Tcl_ExitThread, Tcl_FinalizeThread, Tcl_CreateThreadExitHandler, Tcl_DeleteThreadExitHandler

KEYWORDS

thread, mutex, condition variable, thread local storage

Copyright ' 1999 Scriptics Corporation
Copyright ' 1998 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

Tcl_UniChar, Tcl_UniCharCaseMatch, Tcl_UniCharNcasecmp, Tcl_UniCharToUtf, Tcl_UtfToUniChar,
Tcl_UniCharToUtfDString, Tcl_UtfToUniCharDString, Tcl_UniCharLen, Tcl_UniCharNcmp,
Tcl_UtfCharComplete, Tcl_NumUtfChars, Tcl_UtfFindFirst, Tcl_UtfFindLast, Tcl_UtfNext, Tcl_UtfPrev,
Tcl_UniCharAtIndex, Tcl_UtfAtIndex, Tcl_UtfBackslash – routines for manipulating UTF-8 strings.

SYNOPSIS

#include <tcl.h>
typedef ... Tcl_UniChar;
int
Tcl_UniCharToUtf(ch, buf)
int
Tcl_UtfToUniChar(src, chPtr)
char *
Tcl_UniCharToUtfDString(uniStr, numChars, dstPtr)
Tcl_UniChar *
Tcl_UtfToUniCharDString(src, len, dstPtr)
int
Tcl_UniCharLen(uniStr)
int
Tcl_UniCharNcmp(uniStr, uniStr, num)
int
Tcl_UniCharNcasecmp(uniStr, uniStr, num)
int
Tcl_UniCharCaseMatch(uniStr, uniPattern, nocase)
int
Tcl_UtfNcmp(src, src, num)
int
Tcl_UtfNcasecmp(src, src, num)
int
Tcl_UtfCharComplete(src, len)
int
Tcl_NumUtfChars(src, len)
CONST char *
Tcl_UtfFindFirst(src, ch)
CONST char *
Tcl_UtfFindLast(src, ch)
CONST char *
Tcl_UtfNext(src)
CONST char *
TclUtfPrev(src, start)
TclUniChar
TclUniCharAtIndex(src, index)
CONST char *
TclUtfAtIndex(src, index)
int
TclUtfBackslash(src, readPtr, dst)

ARGUMENTS

char *buf (out)
Buffer in which the UTF–8 representation of the Tcl UniChar is stored. At most TCL_UTF_MAX bytes are stored in the buffer.

int ch (in)
The Tcl UniChar to be converted or examined.

TclUniChar *chPtr (out)
Filled with the Tcl UniChar represented by the head of the UTF–8 string.

CONST char *src (in)
Pointer to a UTF–8 string.

CONST TclUniChar *uniStr (in)
A null–terminated Unicode string.

CONST TclUniChar *uniPattern (in)
A null–terminated Unicode string.

int len (in)
The length of the UTF–8 string in bytes (not UTF–8 characters). If negative, all bytes up to the first null byte are used.

int numChars (in)
The length of the Unicode string in characters. Must be greater than or equal to 0.

TclDString *dstPtr (in/out)
A pointer to a previously–initialized TclDString.

unsigned long num (in)
The number of characters to compare.

CONST char *start (in)
Pointer to the beginning of a UTF–8 string.

int index (in)
The index of a character (not byte) in the UTF–8 string.

int *readPtr (out)
If non–NULL, filled with the number of bytes in the backslash sequence, including the backslash character.

char *dst (out)
Buffer in which the bytes represented by the backslash sequence are stored. At most TCL_UTF_MAX bytes are stored in the buffer.

int nocase (in)
Specifies whether the match should be done case–sensitive (0) or case–insensitive (1).
DESCRIPTION

These routines convert between UTF–8 strings and Tcl_UniChars. A Tcl_UniChar is a Unicode character represented as an unsigned, fixed-size quantity. A UTF–8 character is a Unicode character represented as a varying-length sequence of up to TCL_UTF_MAX bytes. A multibyte UTF–8 sequence consists of a lead byte followed by some number of trail bytes.

TCL_UTF_MAX is the maximum number of bytes that it takes to represent one Unicode character in the UTF–8 representation.

Tcl_UniCharToUtf stores the Tcl_UniChar ch as a UTF–8 string in starting at buf. The return value is the number of bytes stored in buf.

Tcl_UtfToUniChar reads one UTF–8 character starting at src and stores it as a Tcl_UniChar in *chPtr. The return value is the number of bytes read from src.. The caller must ensure that the source buffer is long enough such that this routine does not run off the end and dereference non–existent or random memory; if the source buffer is known to be null–terminated, this will not happen. If the input is not in proper UTF–8 format, Tcl_UtfToUniChar will store the first byte of src in *chPtr as a Tcl_UniChar between 0x0000 and 0x00ff and return 1.

Tcl_UniCharToUtfDString converts the given Unicode string to UTF–8, storing the result in a previously–initialized Tcl_DString. You must specify the length of the given Unicode string. The return value is a pointer to the UTF–8 representation of the Unicode string. Storage for the return value is appended to the end of the Tcl_DString.

Tcl_UtfToUniCharDString converts the given UTF–8 string to Unicode, storing the result in the previously–initialized Tcl_DString, you may either specify the length of the given UTF–8 string or "−1", in which case Tcl_UtfToUniCharDString uses strlen to calculate the length. The return value is a pointer to the Unicode representation of the UTF–8 string. Storage for the return value is appended to the end of the Tcl_DString. The Unicode string is terminated with a Unicode null character.

Tcl_UniCharLen corresponds to strlen for Unicode characters. It accepts a null–terminated Unicode string and returns the number of Unicode characters (not bytes) in that string.

Tcl_UniCharNcmp and Tcl_UniCharNcasecmp correspond to strncmp and strcasecmp, respectively, for Unicode characters. They accepts two null–terminated Unicode strings and the number of characters to compare. Both strings are assumed to be at least len characters long. Tcl_UniCharNcmp compares the two strings character–by–character according to the Unicode character ordering. It returns an integer greater than, equal to, or less than 0 if the first string is greater than, equal to, or less than the second string respectively. Tcl_UniCharNcasecmp is the Unicode case insensitive version.

Tcl_UniCharCaseMatch is the Unicode equivalent to Tcl_StringCaseMatch. It accepts a null–terminated Unicode string, a Unicode pattern, and a boolean value specifying whether the match should be case sensitive and returns whether the string matches the pattern.
Tcl_UtfNcmp corresponds to strncmp for UTF–8 strings. It accepts two null–terminated UTF–8 strings and the number of characters to compare. (Both strings are assumed to be at least len characters long.) Tcl_UtfNcmp compares the two strings character–by–character according to the Unicode character ordering. It returns an integer greater than, equal to, or less than 0 if the first string is greater than, equal to, or less than the second string respectively.

Tcl_UtfNcasecmp corresponds to strncasecmp for UTF–8 strings. It is similar to Tcl_UtfNcmp except comparisons ignore differences in case when comparing upper, lower or title case characters.

Tcl_UtfCharComplete returns 1 if the source UTF–8 string src of length len bytes is long enough to be decoded by Tcl_UtfToUniChar, or 0 otherwise. This function does not guarantee that the UTF–8 string is properly formed. This routine is used by procedures that are operating on a byte at a time and need to know if a full Tcl_UniChar has been seen.

Tcl_NumUtfChars corresponds to strlen for UTF–8 strings. It returns the number of Tcl_UniChars that are represented by the UTF–8 string src. The length of the source string is len bytes. If the length is negative, all bytes up to the first null byte are used.

Tcl_UtfFindFirst corresponds to strchr for UTF–8 strings. It returns a pointer to the first occurrence of the Tcl_UniChar ch in the null–terminated UTF–8 string src. The null terminator is considered part of the UTF–8 string.

Tcl_UtfFindLast corresponds to strrchr for UTF–8 strings. It returns a pointer to the last occurrence of the Tcl_UniChar ch in the null–terminated UTF–8 string src. The null terminator is considered part of the UTF–8 string.

Given src, a pointer to some location in a UTF–8 string, Tcl_UtfNext returns a pointer to the next UTF–8 character in the string. The caller must not ask for the next character after the last character in the string if the string is not terminated by a null character.

Given src, a pointer to some location in a UTF–8 string (or to a null byte immediately following such a string), Tcl_UtfPrev returns a pointer to the closest preceding byte that starts a UTF–8 character. This function will not back up to a position before start, the start of the UTF–8 string. If src was already at start, the return value will be start.

Tcl_UniCharAtIndex corresponds to a C string array dereference or the Pascal Ord() function. It returns the Tcl_UniChar represented at the specified character (not byte) index in the UTF–8 string src. The source string must contain at least index characters. Behavior is undefined if a negative index is given.

Tcl_UtfAtIndex returns a pointer to the specified character (not byte) index in the UTF–8 string src. The source string must contain at least index characters. This is equivalent to calling Tcl_UtfNext index times. If a negative index is given, the return pointer points to the first character in the source string.

Tcl_UtfBackslash is a utility procedure used by several of the Tcl commands. It parses a backslash sequence and stores the properly formed UTF–8 character represented by the backslash sequence in the output buffer.
At most TCL=UTF_MAX bytes are stored in the buffer. Tcl_UtfBackslash modifies *readPtr to contain the number of bytes in the backslash sequence, including the backslash character. The return value is the number of bytes stored in the output buffer.

See the Tcl manual entry for information on the valid backslash sequences. All of the sequences described in the Tcl manual entry are supported by Tcl_UtfBackslash.

**KEYWORDS**

utf, unicode, backslash

---

**Copyright** ' 1997 Sun Microsystems, Inc.
**Copyright** ' 1995−1997 Roger E. Critchlow Jr.

---
<table>
<thead>
<tr>
<th>Tk Colormap</th>
<th>Tk FreeXId</th>
<th>Tk InternalBorderTop</th>
<th>Tk SetInternalBorderEx</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tk ComputeTextLayout</td>
<td>Tk GeometryRequest</td>
<td>Tk IntersectTextLayout</td>
<td>Tk SetMinimumRequestSize</td>
</tr>
<tr>
<td>Tk ConfigureInfo</td>
<td>Tk Get3DBorder</td>
<td>Tk IsContainer</td>
<td>Tk SetOptions</td>
</tr>
<tr>
<td>Tk ConfigureValue</td>
<td>Tk Get3DBorderFromObj</td>
<td>Tk IsEmbedded</td>
<td>Tk SetWindowBackground</td>
</tr>
<tr>
<td>Tk ConfigureWidget</td>
<td>Tk GetAllBindings</td>
<td>Tk IsMapped</td>
<td>Tk SetWindowBackground</td>
</tr>
<tr>
<td>Tk ConfigureWindow</td>
<td>Tk GetAnchor</td>
<td>Tk IsTopLevel</td>
<td>Tk SetWindowBorder</td>
</tr>
<tr>
<td>Tk CoordsToWindow</td>
<td>Tk GetAnchorFromObj</td>
<td>Tk Main</td>
<td>Tk SetWindowBorderPix</td>
</tr>
<tr>
<td>Tk CreateBinding</td>
<td>Tk GetAtomName</td>
<td>Tk MainLoop</td>
<td>Tk SetWindowColormap</td>
</tr>
<tr>
<td>Tk CreateBindingTable</td>
<td>Tk GetBinding</td>
<td>Tk MainWindow</td>
<td>Tk SetWindowVisual</td>
</tr>
<tr>
<td>Tk CreateClientMessageHandler</td>
<td>Tk GetBitmap</td>
<td>Tk MaintainGeometry</td>
<td>Tk SizeOfBitmap</td>
</tr>
<tr>
<td>Tk CreateErrorHandler</td>
<td>Tk GetBitmapFromObj</td>
<td>Tk MakeWindowExist</td>
<td>Tk SizeOfImage</td>
</tr>
<tr>
<td>Tk CreateGenericHandler</td>
<td>Tk GetCapStyle</td>
<td>Tk ManageGeometry</td>
<td>Tk StrictMotif</td>
</tr>
<tr>
<td>Tk CreateImageType</td>
<td>Tk GetColor</td>
<td>Tk MapWindow</td>
<td>Tk TextLayoutToPostscript</td>
</tr>
<tr>
<td>Tk CreateItemType</td>
<td>Tk GetColorByValue</td>
<td>Tk MeasureChars</td>
<td>Tk TextWidth</td>
</tr>
<tr>
<td>Tk CreateOptionTable</td>
<td>Tk GetColorFromObj</td>
<td>Tk MinReqHeight</td>
<td>Tk Uid</td>
</tr>
<tr>
<td>Tk CreatePhotoImageFormat</td>
<td>Tk GetColormap</td>
<td>Tk MinReqWidth</td>
<td>Tk UnderdefineCursor</td>
</tr>
<tr>
<td>Tk CreateSelHandler</td>
<td>Tk GetCursor</td>
<td>Tk MoveResizeWindow</td>
<td>Tk UnderlineChars</td>
</tr>
<tr>
<td>Tk CreateWindow</td>
<td>Tk GetCursorFromData</td>
<td>Tk MoveToplevelWindow</td>
<td>Tk UnderlineTextLayout</td>
</tr>
<tr>
<td>Tk CreateWindowFromPath</td>
<td>Tk GetDash</td>
<td>Tk MoveWindow</td>
<td>Tk Ungrab</td>
</tr>
<tr>
<td>Tk DefineBitmap</td>
<td>Tk GetFont</td>
<td>Tk Name</td>
<td>Tk UnmaintainGeometry</td>
</tr>
<tr>
<td>Tk DefineCursor</td>
<td>Tk GetFontFromObj</td>
<td>Tk NameOf3DBorder</td>
<td>Tk UnmapWindow</td>
</tr>
<tr>
<td>Tk DeleteAllBindings</td>
<td>Tk GetFontMetrics</td>
<td>Tk NameOfAnchor</td>
<td>Tk UnsetGrid</td>
</tr>
<tr>
<td>Tk DeleteBinding</td>
<td>Tk GetGC</td>
<td>Tk NameOfBitmap</td>
<td>Tk Visual</td>
</tr>
<tr>
<td>Tk DeleteBindingTable</td>
<td>Tk GetHINSTANCE</td>
<td>Tk NameOfCapStyle</td>
<td>Tk Width</td>
</tr>
<tr>
<td>Tk DeleteClientMessageHandler</td>
<td>Tk GetHWND</td>
<td>Tk NameOfColor</td>
<td>Tk X</td>
</tr>
<tr>
<td>Tk DeleteErrorHandler</td>
<td>Tk getImage</td>
<td>Tk NameOfCursor</td>
<td>Tk WindowId</td>
</tr>
<tr>
<td>Tk DeleteEventHandler</td>
<td>Tk GetImageMasterData</td>
<td>Tk NameOfFont</td>
<td>Tk Y</td>
</tr>
<tr>
<td>Tk DeleteGenericHandler</td>
<td>Tk GetItemTypes</td>
<td>Tk NameOfImage</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tk NameOfJoinStyle</td>
<td></td>
</tr>
</tbody>
</table>
GetHWND

NAME

Tk_GetHWND, Tk_AttachHWND – manage interaction between the Windows handle and an X window

SYNOPSIS

#include <tkPlatDecls.h>
HWND
Tk_GetHWND(window)
Window
Tk_AttachHWND(tkwin, hwnd)

ARGUMENTS

Window window (in)
   X token for window.
Tk_Window tkwin (in)
   Tk window for window.
HWND hwnd (in)
   Windows HWND for window.

DESCRIPTION

Tk_GetHWND returns the Windows HWND identifier for X Windows window given by window.

Tk_AttachHWND binds the Windows HWND identifier to the specified Tk_Window given by tkwin. It returns an X Windows window that encapsulates the HWND.

KEYWORDS

identifier, window

Copyright ' 1995-1997 Roger E. Critchlow Jr.
AddOption

NAME

Tk_AddOption – Add an option to the option database

SYNOPSIS

ARGUMENTS

DESCRIPTION

This procedure is invoked to add an option to the database associated with tkwin's main window. Name contains the option being specified and consists of names and/or classes separated by asterisks or dots, in the usual X format. Value contains the text string to associate with name; this value will be returned in calls to Tk_GetOption. Priority specifies the priority of the value; when options are queried using Tk_GetOption, the value with the highest priority is returned. Priority must be between 0 and TK_MAX_PRIO. Some common priority values are:
20 Used for default values hard-coded into widgets.
40 Used for options specified in application-specific startup files.
60 Used for options specified in user-specific defaults files, such as .Xdefaults, resource databases loaded into the X server, or user-specific startup files.
80 Used for options specified interactively after the application starts running.

KEYWORDS

class, name, option, add

Copyright © 1995-1997 Roger E. Critchlow Jr.
3DBorder

NAME

Tk_Alloc3DBorderFromObj, Tk_Get3DBorder, Tk_Get3DBorderFromObj, Tk_Draw3DRectangle,
Tk_Fill3DRectangle, Tk_Draw3DPolygon, Tk_Fill3DPolygon, Tk_3DVerticalBevel, Tk_3DHorizontalBevel,
Tk_SetBackgroundColor, Tk_NameOf3DBorder, Tk_3DBorderColor, Tk_3DBorderGC,
Tk_Free3DBorderFromObj, Tk_Free3DBorder – draw borders with three–dimensional appearance

SYNOPSIS

#include <tk.h>
Tk_3DBorder
Tk_Alloc3DBorderFromObj(interp, tkwin, objPtr)
Tk_3DBorder
Tk_Get3DBorder(interp, tkwin, colorName)
Tk_3DBorder
Tk_Get3DBorderFromObj(tkwin, objPtr)
void
 Tk_Draw3DRectangle(tkwin, drawable, border, x, y, width, height, borderWidth, relief)
void
 Tk_Fill3DRectangle(tkwin, drawable, border, x, y, width, height, borderWidth, relief)
void
 Tk_Draw3DPolygon(tkwin, drawable, border, pointPtr, numPoints, polyBorderWidth, leftRelief)
void
 Tk_Fill3DPolygon(tkwin, drawable, border, pointPtr, numPoints, polyBorderWidth, leftRelief)
void
 Tk_3DVerticalBevel(tkwin, drawable, border, x, y, width, height, leftBevel, relief)
void
 Tk_3DHorizontalBevel(tkwin, drawable, border, x, y, width, height, leftIn, rightIn, topBevel, relief)
void
 Tk_SetBackgroundColorFromBorder(tkwin, border)
CONST char *
 Tk_NameOf3DBorder(border)
XColor *
 Tk_3DBorderColor(border)
GC *
 Tk_3DBorderGC(tkwin, border, which)
Tk_Free3DBorderFromObj(tkwin, objPtr)
Tk_Free3DBorder(border)
ARGUMENTS

Tcl_Interp *interp (in)
Interpreter to use for error reporting.

Tk_Window tkwin (in)
Token for window (for all procedures except Tk_Get3DBorder, must be the window for which the border was allocated).

Tcl_Obj *objPtr (in)
Pointer to object whose value describes color corresponding to background (flat areas). Illuminated edges will be brighter than this and shadowed edges will be darker than this.

char *colorName (in)
Same as objPtr except value is supplied as a string rather than an object.

Drawable drawable (in)
X token for window or pixmap; indicates where graphics are to be drawn. Must either be the X window for tkwin or a pixmap with the same screen and depth as tkwin.

Tk_3DBorder border (in)
Token for border previously allocated in call to Tk_Get3DBorder.

int x (in)
X-coordinate of upper-left corner of rectangle describing border or bevel, in pixels.

int y (in)
Y-coordinate of upper-left corner of rectangle describing border or bevel, in pixels.

int width (in)
Width of rectangle describing border or bevel, in pixels.

int height (in)
Height of rectangle describing border or bevel, in pixels.

int borderWidth (in)
Width of border in pixels. Positive means border is inside rectangle given by x, y, width, height, negative means border is outside rectangle.

int relief (in)
Indicates 3-D position of interior of object relative to exterior; should be TK_RELIEF_RAISED, TK_RELIEF_SUNKEN, TK_RELIEF_GROOVE, TK_RELIEF_SOLID, or TK_RELIEF_RIDGE (may also be TK_RELIEF_FLAT for Tk_Fill3DRectangle).

XPoint *pointPtr (in)
Pointer to array of points describing the set of vertices in a polygon. The polygon need not be closed (it will be closed automatically if it isn't).

int numPoints (in)
Number of points at *pointPtr.

int polyBorderWidth (in)
Width of border in pixels. If positive, border is drawn to left of trajectory given by pointPtr; if negative, border is drawn to right of trajectory. If leftRelief is TK_RELIEF_GROOVE or TK_RELIEF_RIDGE then the border is centered on the trajectory.

int leftRelief (in)
Height of left side of polygon's path relative to right. TK_RELIEF_RAISED means left side should appear higher and TK_RELIEF_SUNKEN means right side should appear higher; TK_RELIEF_GROOVE and TK_RELIEF_RIDGE mean the obvious things. For Tk_Fill3DPolygon,
TK_RELIEF_FLAT may also be specified to indicate no difference in height.

- **int leftBevel (in)**
  - Non-zero means this bevel forms the left side of the object; zero means it forms the right side.

- **int leftIn (in)**
  - Non-zero means that the left edge of the horizontal bevel angles in, so that the bottom of the edge is farther to the right than the top. Zero means the edge angles out, so that the bottom is farther to the left than the top.

- **int rightIn (in)**
  - Non-zero means that the right edge of the horizontal bevel angles in, so that the bottom of the edge is farther to the left than the top. Zero means the edge angles out, so that the bottom is farther to the right than the top.

- **int topBevel (in)**
  - Non-zero means this bevel forms the top side of the object; zero means it forms the bottom side.

- **int which (in)**
  - Specifies which of the border's graphics contexts is desired. Must be TK_3D_FLAT_GC, TK_3D_LIGHT_GC, or TK_3D_DARK_GC.

**DESCRIPTION**

These procedures provide facilities for drawing window borders in a way that produces a three−dimensional appearance. `Tk_Alloc3DBorderFromObj` allocates colors and Pixmaps needed to draw a border in the window given by the `tkwin` argument. The value of `objPtr` is a standard Tk color name that determines the border colors. The color indicated by `objPtr` will not actually be used in the border; it indicates the background color for the window (i.e. a color for flat surfaces). The illuminated portions of the border will appear brighter than indicated by `objPtr`, and the shadowed portions of the border will appear darker than `objPtr`.

`Tk_Alloc3DBorderFromObj` returns a token that may be used in later calls to `Tk_Draw3DRectangle`. If an error occurs in allocating information for the border (e.g. a bogus color name was given) then NULL is returned and an error message is left in `interp->result`. If it returns successfully, `Tk_Alloc3DBorderFromObj` caches information about the return value in `objPtr`, which speeds up future calls to `Tk_Alloc3DBorderFromObj` with the same `objPtr` and `tkwin`.

`Tk_Get3DBorder` is identical to `Tk_Alloc3DBorderFromObj` except that the color is specified with a string instead of an object. This prevents `Tk_Get3DBorder` from caching the return value, so `Tk_Get3DBorder` is less efficient than `Tk_Alloc3DBorderFromObj`.

`Tk_Get3DBorderFromObj` returns the token for an existing border, given the window and color name used to create the border. `Tk_Get3DBorderFromObj` doesn't actually create the border; it must already have been created with a previous call to `Tk_Alloc3DBorderFromObj` or `Tk_Get3DBorder`. The return value is cached in `objPtr`, which speeds up future calls to `Tk_Get3DBorderFromObj` with the same `objPtr` and `tkwin`.

Once a border structure has been created, `Tk_Draw3DRectangle` may be invoked to draw the border. The `tkwin` argument specifies the window for which the border was allocated, and `drawable` specifies a window or
pixmap in which the border is to be drawn. *Drawable* need not refer to the same window as *tkwin*, but it must refer to a compatible pixmap or window: one associated with the same screen and with the same depth as *tkwin*. The *x*, *y*, *width*, and *height* arguments define the bounding box of the border region within *drawable* (usually *x* and *y* are zero and *width* and *height* are the dimensions of the window), and *borderWidth* specifies the number of pixels actually occupied by the border. The *relief* argument indicates which of several three-dimensional effects is desired: TK_RELIEF_RAISED means that the interior of the rectangle should appear raised relative to the exterior of the rectangle, and TK_RELIEF_SUNKEN means that the interior should appear depressed. TK_RELIEF_GROOVE and TK_RELIEF_RIDGE mean that there should appear to be a groove or ridge around the exterior of the rectangle.

**Tk_Fill3DRectangle** is somewhat like **Tk_Draw3DRectangle** except that it first fills the rectangular area with the background color (one corresponding to the color used to create *border*). Then it calls **Tk_Draw3DRectangle** to draw a border just inside the outer edge of the rectangular area. The argument *relief* indicates the desired effect (TK_RELIEF_FLAT means no border should be drawn; all that happens is to fill the rectangle with the background color).

The procedure **Tk_Draw3DPolygon** may be used to draw more complex shapes with a three-dimensional appearance. The *pointPtr* and *numPoints* arguments define a trajectory, *polyBorderWidth* indicates how wide the border should be (and on which side of the trajectory to draw it), and *leftRelief* indicates which side of the trajectory should appear raised. **Tk_Draw3DPolygon** draws a border around the given trajectory using the colors from *border* to produce a three-dimensional appearance. If the trajectory is non-self-intersecting, the appearance will be a raised or sunken polygon shape. The trajectory may be self-intersecting, although it's not clear how useful this is.

**Tk_Fill3DPolygon** is to **Tk_Draw3DPolygon** what **Tk_Fill3DRectangle** is to **Tk_Draw3DRectangle**: it fills the polygonal area with the background color from *border*, then calls **Tk_Draw3DPolygon** to draw a border around the area (unless *leftRelief* is TK_RELIEF_FLAT; in this case no border is drawn).

The procedures **Tk_3DVerticalBevel** and **Tk_3DHorizontalBevel** provide lower-level drawing primitives that are used by procedures such as **Tk_Draw3DRectangle**. These procedures are also useful in their own right for drawing rectilinear border shapes. **Tk_3DVerticalBevel** draws a vertical beveled edge, such as the left or right side of a rectangle, and **Tk_3DHorizontalBevel** draws a horizontal beveled edge, such as the top or bottom of a rectangle. Each procedure takes *x*, *y*, *width*, and *height* arguments that describe the rectangular area of the beveled edge (e.g., *width* is the border width for **Tk_3DVerticalBevel**). The *leftBorder* and *topBorder* arguments indicate the position of the border relative to the "inside" of the object, and *relief* indicates the relief of the inside of the object relative to the outside. **Tk_3DVerticalBevel** just draws a rectangular region. **Tk_3DHorizontalBevel** draws a trapezoidal region to generate mitered corners; it should be called after **Tk_3DVerticalBevel** (otherwise **Tk_3DVerticalBevel** will overwrite the mitering in the corner). The *leftIn* and *rightIn* arguments to **Tk_3DHorizontalBevel** describe the mitering at the corners; a value of 1 means that the bottom edge of the trapezoid will be shorter than the top, 0 means it will be longer. For example, to draw a rectangular border the top bevel should be drawn with 1 for both *leftIn* and *rightIn*, and the bottom bevel should be drawn with 0 for both arguments.

The procedure **Tk_SetBackgroundFromBorder** will modify the background pixel and/or pixmap of *tkwin* to produce a result compatible with *border*. For color displays, the resulting background will just be the color
specified when \textit{border} was created; for monochrome displays, the resulting background will be a light stipple pattern, in order to distinguish the background from the illuminated portion of the border.

Given a token for a border, the procedure \texttt{Tk\_NameOf3DBorder} will return the color name that was used to create the border.

The procedure \texttt{Tk\_3DBorderColor} returns the XColor structure that will be used for flat surfaces drawn for its \textit{border} argument by procedures like \texttt{Tk\_Fill3DRectangle}. The return value corresponds to the color name that was used to create the border. The XColor, and its associated pixel value, will remain allocated as long as \textit{border} exists.

The procedure \texttt{Tk\_3DBorderGC} returns one of the X graphics contexts that are used to draw the border. The argument \textit{which} selects which one of the three possible GC's: \texttt{TK\_3D\_FLAT\_GC} returns the context used for flat surfaces, \texttt{TK\_3D\_LIGHT\_GC} returns the context for light shadows, and \texttt{TK\_3D\_DARK\_GC} returns the context for dark shadows.

When a border is no longer needed, \texttt{Tk\_Free3DBorderFromObj} or \texttt{Tk\_Free3DBorder} should be called to release the resources associated with it. For \texttt{Tk\_Free3DBorderFromObj} the border to release is specified with the window and color name used to create the border; for \texttt{Tk\_Free3DBorder} the border to release is specified with the Tk\_3DBorder token for the border. There should be exactly one call to \texttt{Tk\_Free3DBorderFromObj} or \texttt{Tk\_Free3DBorder} for each call to \texttt{Tk\_Alloc3DBorderFromObj} or \texttt{Tk\_Get3DBorder}.

\section*{KEYWORDS}

3D, background, border, color, depressed, illumination, object, polygon, raised, shadow, three–dimensional effect

\textbf{Copyright} \textsuperscript{*} 1990–1993 The Regents of the University of California.
\textbf{Copyright} \textsuperscript{*} 1994–1998 Sun Microsystems, Inc.
\textbf{Copyright} \textsuperscript{*} 1995–1997 Roger E. Critchlow Jr.
GetBitmap

NAME

Tk_AllocBitmapFromObj, Tk_GetBitmap, Tk_GetBitmapFromObj, Tk_DefineBitmap,
Tk_NameOfBitmap, Tk_SizeOfBitmap, Tk_FreeBitmapFromObj, Tk_FreeBitmap –
maintain database of single-plane pixmaps

SYNOPSIS

ARGUMENTS

DESCRIPTION

@fileName
name
error
gray75
gray50
gray25
gray12
hourglass
info
questhead
question
warning
document
stationery
edition
application
accessory
folder
pfolder
trash
floppy
ramdisk
cdrom
preferences
querydoc
stop
note
caution

BUGS

KEYWORDS
NAME

Tk_AllocBitmapFromObj, Tk_GetBitmap, Tk_GetBitmapFromObj, Tk_DefineBitmap, Tk_NameOfBitmap, Tk_SizeOfBitmap, Tk_FreeBitmapFromObj, Tk_FreeBitmap – maintain database of single-plane pixmaps

SYNOPSIS

#include <tk.h>

Pixmap Tk_GetBitmapFromObj(interp, tkwin, objPtr)
Pixmap Tk_GetBitmap(interp, tkwin, info)
Pixmap Tk_GetBitmapFromObj(tkwin, objPtr)
int Tk_DefineBitmap(interp, name, source, width, height)
CONST char * Tk_NameOfBitmap(display, bitmap)
Tk_SizeOfBitmap(display, bitmap, widthPtr, heightPtr)
Tk_FreeBitmapFromObj(tkwin, objPtr)
Tk_FreeBitmap(display, bitmap)

ARGUMENTS

Tcl_Interp *interp (in)
Interpreter to use for error reporting; if NULL then no error message is left after errors.

Tk_Window tkwin (in)
Token for window in which the bitmap will be used.

Tcl_Obj *objPtr (in/out)
String value describes desired bitmap; internal rep will be modified to cache pointer to corresponding Pixmap.

CONST char *info (in)
Same as objPtr except description of bitmap is passed as a string and resulting Pixmap isn't cached.

CONST char *name (in)
Name for new bitmap to be defined.

CONST char *source (in)
Data for bitmap, in standard bitmap format. Must be stored in static memory whose value will never change.

int width (in)
Width of bitmap.

int height (in)
Height of bitmap.

int *widthPtr (out)
Pointer to word to fill in with bitmap's width.

int *heightPtr (out)
Pointer to word to fill in with bitmap's height.

Display *display (in)
Display for which bitmap was allocated.

Pixmap bitmap (in)
Identifier for a bitmap allocated by \texttt{TkAllocBitmapFromObj} or \texttt{TkGetBitmap}.

\section*{DESCRIPTION}

These procedures manage a collection of bitmaps (one−plane pixmaps) being used by an application. The procedures allow bitmaps to be re−used efficiently, thereby avoiding server overhead, and also allow bitmaps to be named with character strings.

\texttt{TkAllocBitmapFromObj} returns a Pixmap identifier for a bitmap that matches the description in \texttt{objPtr} and is suitable for use in \texttt{tkwin}. It re−uses an existing bitmap, if possible, and creates a new one otherwise. \texttt{ObjPtr}'s value must have one of the following forms:

\begin{itemize}
  \item \texttt{@fileName}

    \textit{FileName} must be the name of a file containing a bitmap description in the standard X11 or X10 format.

  \item \texttt{name}

    \textit{Name} must be the name of a bitmap defined previously with a call to \texttt{TkDefineBitmap}. The following names are pre−defined by Tk:

    \begin{itemize}
      \item \texttt{error}

        The international "don't" symbol: a circle with a diagonal line across it.

      \item \texttt{gray75}

        75\% gray: a checkerboard pattern where three out of four bits are on.

      \item \texttt{gray50}

        50\% gray: a checkerboard pattern where every other bit is on.

      \item \texttt{gray25}

        25\% gray: a checkerboard pattern where one out of every four bits is on.

      \item \texttt{gray12}

        12.5\% gray: a pattern where one−eighth of the bits are on, consisting of every fourth pixel in every other row.

      \item \texttt{hourglass}

        An hourglass symbol.

      \item \texttt{info}

        A large letter ``i''.

      \item \texttt{questhead}

        The silhouette of a human head, with a question mark in it.

      \item \texttt{question}

        A large question−mark.

      \item \texttt{warning}

        A large exclamation point.

    \end{itemize}

In addition, the following pre−defined names are available only on the \texttt{Macintosh} platform:
document
  A generic document.
stationery
  Document stationery.
edition
  The edition symbol.
application
  Generic application icon.
accessory
  A desk accessory.
folder
  Generic folder icon.
pfolder
  A locked folder.
trash
  A trash can.
floppy
  A floppy disk.
ramdisk
  A floppy disk with chip.
cdrom
  A cd disk icon.
preferences
  A folder with prefs symbol.
querydoc
  A database document icon.
stop
  A stop sign.
note
  A face with balloon words.
caution
  A triangle with an exclamation point.

Under normal conditions, Tk_AllocBitmapFromObj returns an identifier for the requested bitmap. If an error occurs in creating the bitmap, such as when objPtr refers to a non-existent file, then None is returned and an error message is left in interp’s result if interp isn’t NULL. Tk_AllocBitmapFromObj caches information about the return value in objPtr, which speeds up future calls to procedures such as Tk_AllocBitmapFromObj and Tk_GetBitmapFromObj.

Tk_GetBitmap is identical to Tk_AllocBitmapFromObj except that the description of the bitmap is specified with a string instead of an object. This prevents Tk_GetBitmap from caching the return value, so Tk_GetBitmap is less efficient than Tk_AllocBitmapFromObj.

Tk_GetBitmapFromObj returns the token for an existing bitmap, given the window and description used to create the bitmap. Tk_GetBitmapFromObj doesn’t actually create the bitmap; the bitmap must already have
been created with a previous call to `Tk_AllocBitmapFromObj` or `Tk_GetBitmap`. The return value is cached in `objPtr`, which speeds up future calls to `Tk_GetBitmapFromObj` with the same `objPtr` and `tkwin`.

`Tk_DefineBitmap` associates a name with in-memory bitmap data so that the name can be used in later calls to `Tk_AllocBitmapFromObj` or `Tk_GetBitmap`. The `nameId` argument gives a name for the bitmap; it must not previously have been used in a call to `Tk_DefineBitmap`. The arguments `source`, `width`, and `height` describe the bitmap. `Tk_DefineBitmap` normally returns TCL_OK; if an error occurs (e.g. a bitmap named `nameId` has already been defined) then TCL_ERROR is returned and an error message is left in `interp->result`. Note: `Tk_DefineBitmap` expects the memory pointed to by `source` to be static: `Tk_DefineBitmap` doesn't make a private copy of this memory, but uses the bytes pointed to by `source` later in calls to `Tk_AllocBitmapFromObj` or `Tk_GetBitmap`.

Typically `Tk_DefineBitmap` is used by `#include`-ing a bitmap file directly into a C program and then referencing the variables defined by the file. For example, suppose there exists a file `stip.bitmap`, which was created by the `bitmap` program and contains a stipple pattern. The following code uses `Tk_DefineBitmap` to define a new bitmap named `foo`:

```c
Pixmap bitmap;
#include "stip.bitmap"
Tk_DefineBitmap(interp, "foo", stip_bits, stip_width, stip_height);
... bitmap = Tk_GetBitmap(interp, tkwin, "foo");
```

This code causes the bitmap file to be read at compile-time and incorporates the bitmap information into the program's executable image. The same bitmap file could be read at run-time using `Tk_GetBitmap`:

```c
Pixmap bitmap;
bitmap = Tk_GetBitmap(interp, tkwin, @stip.bitmap);
```

The second form is a bit more flexible (the file could be modified after the program has been compiled, or a different string could be provided to read a different file), but it is a little slower and requires the bitmap file to exist separately from the program.

Tk maintains a database of all the bitmaps that are currently in use. Whenever possible, it will return an existing bitmap rather than creating a new one. When a bitmap is no longer used, Tk will release it automatically. This approach can substantially reduce server overhead, so `Tk_AllocBitmapFromObj` and `Tk_GetBitmap` should generally be used in preference to Xlib procedures like `XReadBitmapFile`.

The bitmaps returned by `Tk_AllocBitmapFromObj` and `Tk_GetBitmap` are shared, so callers should never modify them. If a bitmap must be modified dynamically, then it should be created by calling Xlib procedures such as `XReadBitmapFile` or `XCreatePixmap` directly.

The procedure `Tk_NameOfBitmap` is roughly the inverse of `Tk_GetBitmap`. Given an X Pixmap argument, it returns the textual description that was passed to `Tk_GetBitmap` when the bitmap was created. `Bitmap` must have been the return value from a previous call to `Tk_AllocBitmapFromObj` or `Tk_GetBitmap`.
Tk_SizeOfBitmap returns the dimensions of its bitmap argument in the words pointed to by the widthPtr and heightPtr arguments. As with Tk_NameOfBitmap, bitmap must have been created by Tk_AllocBitmapFromObj or Tk_GetBitmap.

When a bitmap is no longer needed, Tk_FreeBitmapFromObj or Tk_FreeBitmap should be called to release it. For Tk_FreeBitmapFromObj the bitmap to release is specified with the same information used to create it; for Tk_FreeBitmap the bitmap to release is specified with its Pixmap token. There should be exactly one call to Tk_FreeBitmapFromObj or Tk_FreeBitmap for each call to Tk_AllocBitmapFromObj or Tk_GetBitmap.

**BUGS**

In determining whether an existing bitmap can be used to satisfy a new request, Tk_AllocBitmapFromObj and Tk_GetBitmap consider only the immediate value of the string description. For example, when a file name is passed to Tk_GetBitmap, Tk_GetBitmap will assume it is safe to re-use an existing bitmap created from the same file name: it will not check to see whether the file itself has changed, or whether the current directory has changed, thereby causing the name to refer to a different file.

**KEYWORDS**

bitmap, pixmap

---

*Copyright © 1990 The Regents of the University of California.*

*Copyright © 1994-1998 Sun Microsystems, Inc.*

*Copyright © 1995-1997 Roger E. Critchlow Jr.*
GetColor

NAME

Tk_AllocColorFromObj, Tk_GetColor, Tk_GetColorFromObj, Tk_GetColorByValue, Tk_NameOfColor, Tk_FreeColorFromObj, Tk_FreeColor – maintain database of colors

SYNOPSIS

ARGUMENTS

colorme
#RGB
#RRGGBB
#RRRGGGGBBB
#RRRRGGGGGBBB

KEYWORDS

NAME

Tk_AllocColorFromObj, Tk_GetColor, Tk_GetColorFromObj, Tk_GetColorByValue, Tk_NameOfColor, Tk_FreeColorFromObj, Tk_FreeColor – maintain database of colors

SYNOPSIS

#include <tk.h>
XColor *
Tk_AllocColorFromObj(interp, tkwin, objPtr)
XColor *
Tk_GetColor(interp, tkwin, name)
XColor *
Tk_GetColorFromObj(tkwin, objPtr)
XColor *
Tk_GetColorByValue(tkwin, prefPtr)
CONST char *
Tk_NameOfColor(colorPtr)
GC
Tk_GCForColor(colorPtr, drawable)
Tk_FreeColorFromObj(tkwin, objPtr)
Tk_FreeColor(colorPtr)

ARGUMENTS

Tcl_Interp *interp (in)

Interpreter to use for error reporting.
Tk_Window tkwin (in)
  Token for window in which color will be used.
Tcl_Obj *objPtr (in/out)
  String value describes desired color; internal rep will be modified to cache pointer to corresponding
  (XColor *).
char *name (in)
  Same as objPtr except description of color is passed as a string and resulting (XColor *) isn't cached.
XColor *prefPtr (in)
  Indicates red, green, and blue intensities of desired color.
XColor *colorPtr (in)
  Pointer to X color information. Must have been allocated by previous call to
  Tk_AllocColorFromObj, Tk_GetColor or Tk_GetColorByValue, except when passed to
  Tk_NameOfColor.
Drawable drawable (in)
  Drawable in which the result graphics context will be used. Must have same screen and depth as the
  window for which the color was allocated.

DESCRIPTION

These procedures manage the colors being used by a Tk application. They allow colors to be shared whenever
possible, so that colormap space is preserved, and they pick closest available colors when colormap space is
exhausted.

Given a textual description of a color, Tk_AllocColorFromObj locates a pixel value that may be used to
render the color in a particular window. The desired color is specified with an object whose string value must
have one of the following forms:

colorname
  Any of the valid textual names for a color defined in the server's color database file, such as red or
  PeachPuff.
#RGB
#RRGGBB
#RRRGGGBBB
#RRRRGGGGGBBBB
  A numeric specification of the red, green, and blue intensities to use to display the color. Each R, G,
  or B represents a single hexadecimal digit. The four forms permit colors to be specified with 4-bit,
  8-bit, 12-bit or 16-bit values. When fewer than 16 bits are provided for each color, they represent
  the most significant bits of the color. For example, #3a7 is the same as #3000a0007000.

Tk_AllocColorFromObj returns a pointer to an XColor structure; the structure indicates the exact intensities
of the allocated color (which may differ slightly from those requested, depending on the limitations of the
screen) and a pixel value that may be used to draw with the color in tkwin. If an error occurs in
Tk_AllocColorFromObj (such as an unknown color name) then NULL is returned and an error message is
stored in interp's result if interp isn't NULL. If the colormap for tkwin is full, Tk_AllocColorFromObj will
use the closest existing color in the colormap. Tk_AllocColorFromObj caches information about the return
value in objPtr, which speeds up future calls to procedures such as Tk_AllocColorFromObj and Tk_GetColorFromObj.

Tk_GetColor is identical to Tk_AllocColorFromObj except that the description of the color is specified with a string instead of an object. This prevents Tk_GetColor from caching the return value, so Tk_GetColor is less efficient than Tk_AllocColorFromObj.

Tk_GetColorFromObj returns the token for an existing color, given the window and description used to create the color. Tk_GetColorFromObj doesn't actually create the color; the color must already have been created with a previous call to Tk_AllocColorFromObj or Tk_GetColor. The return value is cached in objPtr, which speeds up future calls to Tk_GetColorFromObj with the same objPtr and tkwin.

Tk_GetColorByValue is similar to Tk_GetColor except that the desired color is indicated with the red, green, and blue fields of the structure pointed to by colorPtr.

This package maintains a database of all the colors currently in use. If the same color is requested multiple times from Tk_GetColor or Tk_AllocColorFromObj (e.g. by different windows), or if the same intensities are requested multiple times from Tk_GetColorByValue, then existing pixel values will be re-used. Re-using an existing pixel avoids any interaction with the window server, which makes the allocation much more efficient. These procedures also provide a portable interface that works across all platforms. For this reason, you should generally use Tk_AllocColorFromObj, Tk_GetColor, or Tk_GetColorByValue instead of lower level procedures like XAllocColor.

Since different calls to this package may return the same shared pixel value, callers should never change the color of a pixel returned by the procedures. If you need to change a color value dynamically, you should use XAllocColorCells to allocate the pixel value for the color.

The procedure Tk_NameOfColor is roughly the inverse of Tk_GetColor. If its colorPtr argument was created by Tk_AllocColorFromObj or Tk_GetColor then the return value is the string that was used to create the color. If colorPtr was created by a call to Tk_GetColorByValue, or by any other mechanism, then the return value is a string that could be passed to Tk_GetColor to return the same color. Note: the string returned by Tk_NameOfColor is only guaranteed to persist until the next call to Tk_NameOfColor.

Tk_GCForColor returns a graphics context whose foreground field is the pixel allocated for colorPtr and whose other fields all have default values. This provides an easy way to do basic drawing with a color. The graphics context is cached with the color and will exist only as long as colorPtr exists; it is freed when the last reference to colorPtr is freed by calling Tk_FreeColor.

When a color is no longer needed Tk_FreeColorFromObj or Tk_FreeColor should be called to release it. For Tk_FreeColorFromObj the color to release is specified with the same information used to create it; for Tk_FreeColor the color to release is specified with a pointer to its XColor structure. There should be exactly one call to Tk_FreeColorFromObj or Tk_FreeColor for each call to Tk_AllocColorFromObj, Tk_GetColor, or Tk_GetColorByValue.
KEYWORDS

color, intensity, object, pixel value

Copyright ' 1990-1991 The Regents of the University of California.
Copyright ' 1994-1998 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
GetCursor

NAME

Tk_AllocCursorFromObj, Tk_GetCursor, Tk_GetCursorFromObj, Tk_GetCursorFromData,
Tk_NameOfCursor, Tk_FreeCursorFromObj, Tk_FreeCursor – maintain database of cursors

SYNOPSIS

ARGUMENTS

DESCRIPTION

name [fgColor [bgColor]]
@sourceName maskName fgColor bgColor
@sourceName fgColor
@sourceName

BUGS

KEYWORDS

NAME

Tk_AllocCursorFromObj, Tk_GetCursor, Tk_GetCursorFromObj, Tk_GetCursorFromData,
Tk_NameOfCursor, Tk_FreeCursorFromObj, Tk_FreeCursor – maintain database of cursors

SYNOPSIS

#include <tk.h>

Tk_Cursor
Tk_AllocCursorFromObj(interp, tkwin, objPtr)
Tk_Cursor
Tk_GetCursor(interp, tkwin, name)
Tk_Cursor
Tk_GetCursorFromObj(tkwin, objPtr)
Tk_Cursor
Tk_GetCursorFromData(interp, tkwin, source, mask, width, height, xHot, yHot, fg, bg)
CONST char *
Tk_NameOfCursor(display, cursor)
Tk_FreeCursorFromObj(tkwin, objPtr)
Tk_FreeCursor(display, cursor)

ARGUMENTS

Tcl_Interp *interp (in)
Interpreter to use for error reporting.
Tk_Window tkwin (in)
Token for window in which the cursor will be used.
**DESCRIPTION**

These procedures manage a collection of cursors being used by an application. The procedures allow cursors to be re-used efficiently, thereby avoiding server overhead, and also allow cursors to be named with character strings.

**Tk_AllocCursorFromObj** takes as argument an object describing a cursor, and returns an opaque Tk identifier for a cursor corresponding to the description. It re-uses an existing cursor if possible and creates a new one otherwise. **Tk_AllocCursorFromObj** caches information about the return value in *objPtr*, which speeds up future calls to procedures such as **Tk_AllocCursorFromObj** and **Tk_GetCursorFromObj**. If an error occurs in creating the cursor, such as when *objPtr* refers to a non-existent file, then **None** is returned and an error message will be stored in *interp*'s result if *interp* isn't NULL. *ObjPtr* must contain a standard Tcl list with one of the following forms:

```
name [fgColor [bgColor]]
```

**name** is the name of a cursor in the standard X cursor cursor, i.e., any of the names defined in `cursorcursor.h`, without the `XC_`. Some example values are `X_cursor`, `hand2`, or `left_ptr`. Appendix B of "The X Window System" by Scheifler & Gettys has illustrations showing what each of these...
The Macintosh version of Tk supports all of the X cursors and will also accept any of the standard Mac cursors including ibeam, crosshair, watch, plus, and arrow. In addition, Tk will load Macintosh cursor resources of the types crsr (color) and CURS (black and white) by the name of the resource. The application and all its open dynamic library's resource files will be searched for the named cursor. If there are conflicts color cursors will always be loaded in preference to black and white cursors.

@sourceName maskName fgColor bgColor

In this form, sourceName and maskName are the names of files describing cursors for the cursor's source bits and mask. Each file must be in standard X11 or X10 cursor format. FgColor and bgColor indicate the colors to use for the cursor, in any of the forms acceptable to Tk_GetColor. This form of the command will not work on Macintosh or Windows computers.

@sourceName fgColor

This form is similar to the one above, except that the source is used as mask also. This means that the cursor's background is transparent. This form of the command will not work on Macintosh or Windows computers.

@sourceName

This form only works on Windows, and will load a Windows system cursor (.ani or .cur) from the file specified in sourceName.

Tk_GetCursor is identical to Tk_AllocCursorFromObj except that the description of the cursor is specified with a string instead of an object. This prevents Tk_GetCursor from caching the return value, so Tk_GetCursor is less efficient than Tk_AllocCursorFromObj.

Tk_GetCursorFromObj returns the token for an existing cursor, given the window and description used to create the cursor. Tk_GetCursorFromObj doesn't actually create the cursor; the cursor must already have been created with a previous call to Tk_AllocCursorFromObj or Tk_GetCursor. The return value is cached in objPtr, which speeds up future calls to Tk_GetCursorFromObj with the same objPtr and tkwin.

Tk_GetCursorFromData allows cursors to be created from in–memory descriptions of their source and mask cursors. Source points to standard cursor data for the cursor's source bits, and mask points to standard cursor data describing which pixels of source are to be drawn and which are to be considered transparent. Width and height give the dimensions of the cursor, xHot and yHot indicate the location of the cursor's hot–spot (the point that is reported when an event occurs), and fg and bg describe the cursor's foreground and background colors textually (any of the forms suitable for Tk_GetColor may be used). Typically, the arguments to Tk_GetCursorFromData are created by including a cursor file directly into the source code for a program, as in the following example:

Tk_Cursor cursor;
#include "source.cursor"
#include "mask.cursor"
cursor = Tk_GetCursorFromData(interp, tkwin, source_bits, mask_bits, source_width, source_height, source_x_hot, source_y_hot, Tk_GetUid("red"), Tk_GetUid("blue"));

Under normal conditions Tk_GetCursorFromData will return an identifier for the requested cursor. If an error occurs in creating the cursor then None is returned and an error message will be stored in interp's result.

Tk_AllocCursorFromObj, Tk_GetCursor, and Tk_GetCursorFromData maintain a database of all the cursors they have created. Whenever possible, a call to Tk_AllocCursorFromObj, Tk_GetCursor, or Tk_GetCursorFromData will return an existing cursor rather than creating a new one. This approach can substantially reduce server overhead, so the Tk procedures should generally be used in preference to Xlib procedures like XCreateFontCursor or XCreatePixmapCursor, which create a new cursor on each call. The Tk procedures are also more portable than the lower−level X procedures.

The procedure Tk_NameOfCursor is roughly the inverse of Tk_GetCursor. If its cursor argument was created by Tk_GetCursor, then the return value is the name argument that was passed to Tk_GetCursor to create the cursor. If cursor was created by a call to Tk_GetCursorFromData, or by any other mechanism, then the return value is a hexadecimal string giving the X identifier for the cursor. Note: the string returned by Tk_NameOfCursor is only guaranteed to persist until the next call to Tk_NameOfCursor. Also, this call is not portable except for cursors returned by Tk_GetCursor.

When a cursor returned by Tk_AllocCursorFromObj, Tk_GetCursor, or Tk_GetCursorFromData is no longer needed, Tk_FreeCursorFromObj or Tk_FreeCursor should be called to release it. For Tk_FreeCursorFromObj the cursor to release is specified with the same information used to create it; for Tk_FreeCursor the cursor to release is specified with its Tk_Cursor token. There should be exactly one call to Tk_FreeCursor for each call to Tk_AllocCursorFromObj, Tk_GetCursor, or Tk_GetCursorFromData.

BUGS

In determining whether an existing cursor can be used to satisfy a new request, Tk_AllocCursorFromObj, Tk_GetCursor, and Tk_GetCursorFromData consider only the immediate values of their arguments. For example, when a file name is passed to Tk_GetCursor, Tk_GetCursor will assume it is safe to re−use an existing cursor created from the same file name: it will not check to see whether the file itself has changed, or whether the current directory has changed, thereby causing the name to refer to a different file. Similarly, Tk_GetCursorFromData assumes that if the same source pointer is used in two different calls, then the pointers refer to the same data; it does not check to see if the actual data values have changed.

KEYWORDS

cursor

Copyright ’ 1990 The Regents of the University of California.
Copyright ’ 1994−1998 Sun Microsystems, Inc.
Copyright ’ 1995−1997 Roger E. Critchlow Jr.
GetFont

NAME

Tk_AllocFontFromObj, Tk_GetFont, Tk_GetFontFromObj, Tk_NameOfFont, Tk_FREEFontFromObj, Tk_FREEFont − maintain database of fonts

SYNOPSIS

#include <tk.h>

Tk_Font
Tk_AllocFontFromObj(interp, tkwin, objPtr)
Tk_Font
Tk_GetFont(interp, tkwin, string)
Tk_Font
Tk_GetFontFromObj(tkwin, objPtr)
CONST char *
Tk_NameOfFont(tkfont)
Tk_Font
Tk_FreeFontFromObj(tkwin, objPtr)
void
Tk_FreeFont(tkfont)

ARGUMENTS

Tcl_Interp *interp (in)
Interpreter to use for error reporting. If NULL, then no error messages are left after errors.

Tk_Window tkwin (in)
Token for window in which font will be used.

Tcl_Obj *objPtr (in/out)
Gives name or description of font. See documentation for the font command for details on acceptable formats. Internal rep will be modified to cache corresponding Tk_Font.

const char *string (in)
Same as objPtr except description of font is passed as a string and resulting Tk_Font isn’t cached.

Tk_Font tkfont (in)
Opaque font token.

DESCRIPTION

Tk_AllocFontFromObj finds the font indicated by objPtr and returns a token that represents the font. The return value can be used in subsequent calls to procedures such as Tk_GetFontMetrics, Tk MeasureChars, and Tk_FreeFont. The Tk_Font token will remain valid until Tk_FreeFontFromObj or Tk_FreeFont is called to release it. ObjPtr can contain either a symbolic name or a font description; see the documentation for
the `font` command for a description of the valid formats. If `Tk_AllocFontFromObj` is unsuccessful (because, for example, `objPtr` did not contain a valid font specification) then it returns `NULL` and leaves an error message in `interp`'s result if `interp` isn't NULL. `Tk_AllocFontFromObj` caches information about the return value in `objPtr`, which speeds up future calls to procedures such as `Tk_AllocFontFromObj` and `Tk_GetFontFromObj`.

`Tk_GetFont` is identical to `Tk_AllocFontFromObj` except that the description of the font is specified with a string instead of an object. This prevents `Tk_GetFont` from caching the matching `Tk_Font`, so `Tk_GetFont` is less efficient than `Tk_AllocFontFromObj`.

`Tk_GetFontFromObj` returns the token for an existing font, given the window and description used to create the font. `Tk_GetFontFromObj` doesn't actually create the font; the font must already have been created with a previous call to `Tk_AllocFontFromObj` or `Tk_GetFont`. The return value is cached in `objPtr`, which speeds up future calls to `Tk_GetFontFromObj` with the same `objPtr` and `tkwin`.

`Tk_AllocFontFromObj` and `Tk_GetFont` maintain a database of all fonts they have allocated. If the same font is requested multiple times (e.g. by different windows or for different purposes), then a single `Tk_Font` will be shared for all uses. The underlying resources will be freed automatically when no-one is using the font anymore.

The procedure `Tk_NameOfFont` is roughly the inverse of `Tk_GetFont`. Given a `tkfont` that was created by `Tk_GetFont` (or `Tk_AllocFontFromObj`), the return value is the string argument that was passed to `Tk_GetFont` to create the font. The string returned by `Tk_NameOfFont` is only guaranteed to persist until the `tkfont` is deleted. The caller must not modify this string.

When a font is no longer needed, `Tk_FreeFontFromObj` or `Tk_FreeFont` should be called to release it. For `Tk_FreeFontFromObj` the font to release is specified with the same information used to create it; for `Tk_FreeFont` the font to release is specified with its `Tk_Font` token. There should be exactly one call to `Tk_FreeFontFromObj` or `Tk_FreeFont` for each call to `Tk_AllocFontFromObj` or `Tk_GetFont`.

**SEE ALSO**

`Tk_FontId`

**KEYWORDS**

font
CanvPsY

NAME

Tk_CanvasPsY, Tk_CanvasPsBitmap, Tk_CanvasPsColor, Tk_CanvasPsFont, Tk_CanvasPsPath, Tk_CanvasPsStipple – utility procedures for generating Postscript for canvases

SYNOPSIS

#include <tk.h>

double
Tk_CanvasPsY(canvas, canvasY)

int
Tk_CanvasPsBitmap(interp, canvas, bitmap, x, y, width, height)

int
Tk_CanvasPsColor(interp, canvas, colorPtr)

int
Tk_CanvasPsFont(interp, canvas, tkFont)

Tk_CanvasPsPath(interp, canvas, coordPtr, numPoints)

int
Tk_CanvasPsStipple(interp, canvas, bitmap)

ARGUMENTS

Tk_Canvas canvas (in)
A token that identifies a canvas widget for which Postscript is being generated.

double canvasY (in)
Y-coordinate in the space of the canvas.

Tcl_Interp * interp (in/out)
A Tcl interpreter; Postscript is appended to its result, or the result may be replaced with an error message.

Pixmap bitmap (in)
Bitmap to use for generating Postscript.

int x (in)
X-coordinate within bitmap of left edge of region to output.

int y (in)
Y-coordinate within bitmap of top edge of region to output.

int width (in)
Width of region of bitmap to output, in pixels.

int height (in)
Height of region of bitmap to output, in pixels.

XColor * colorPtr (in)
Information about color value to set in Postscript.

Tk_Font tkFont (in)
Font for which Postscript is to be generated.

```c
double *coordPtr (in)
```

Pointer to an array of coordinates for one or more points specified in canvas coordinates. The order of values in `coordPtr` is `x1, y1, x2, y2, x3, y3, and so on.

```c
int numPoints (in)
```

Number of points at `coordPtr`.

**DESCRIPTION**

These procedures are called by canvas type managers to carry out common functions related to generating Postscript. Most of the procedures take a `canvas` argument, which refers to a canvas widget for which Postscript is being generated.

- **Tk_CanvasPsY** takes as argument a y-coordinate in the space of a canvas and returns the value that should be used for that point in the Postscript currently being generated for `canvas`. Y coordinates require transformation because Postscript uses an origin at the lower-left corner whereas X uses an origin at the upper-left corner. Canvas x coordinates can be used directly in Postscript without transformation.

- **Tk_CanvasPsBitmap** generates Postscript to describe a region of a bitmap. The Postscript is generated in proper image data format for Postscript, i.e., as data between angle brackets, one bit per pixel. The Postscript is appended to `interp->result` and TCL_OK is returned unless an error occurs, in which case TCL_ERROR is returned and `interp->result` is overwritten with an error message.

- **Tk_CanvasPsColor** generates Postscript to set the current color to correspond to its `colorPtr` argument, taking into account any color map specified in the `postscript` command. It appends the Postscript to `interp->result` and returns TCL_OK unless an error occurs, in which case TCL_ERROR is returned and `interp->result` is overwritten with an error message.

- **Tk_CanvasPsFont** generates Postscript that sets the current font to match `tkFont` as closely as possible. **Tk_CanvasPsFont** takes into account any font map specified in the `postscript` command, and it does the best it can at mapping X fonts to Postscript fonts. It appends the Postscript to `interp->result` and returns TCL_OK unless an error occurs, in which case TCL_ERROR is returned and `interp->result` is overwritten with an error message.

- **Tk_CanvasPsPath** generates Postscript to set the current path to the set of points given by `coordPtr` and `numPoints`. It appends the resulting Postscript to `interp->result`.

- **Tk_CanvasPsStipple** generates Postscript that will fill the current path in stippled fashion. It uses `bitmap` as the stipple pattern and the current Postscript color; ones in the stipple bitmap are drawn in the current color, and zeroes are not drawn at all. The Postscript is appended to `interp->result` and TCL_OK is returned, unless an error occurs, in which case TCL_ERROR is returned and `interp->result` is overwritten with an error message.
KEYWORDS

bitmap, canvas, color, font, path, Postscript, stipple

Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
CanvTxtInfo

NAME

Tk_CanvasTextInfo – additional information for managing text items in canvases

SYNOPSIS

#include <tk.h>
Tk_CanvasTextInfo *
Tk_CanvasGetTextInfo(canvas)

ARGUMENTS

Tk_Canvas canvas (in)
A token that identifies a particular canvas widget.

DESCRIPTION

Textual canvas items are somewhat more complicated to manage than other items, due to things like the
selection and the input focus. Tk_CanvasGetTextInfo may be invoked by a type manager to obtain
additional information needed for items that display text. The return value from Tk_CanvasGetTextInfo is
a pointer to a structure that is shared between Tk and all the items that display text. The structure has the
following form:

typedef struct Tk_CanvasTextInfo {
    Tk_3DBorder selBorder;
    selBorderWidth;
    XColor *selFgColorPtr;
    Tk_Item *selItemPtr;
    int selectFirst;
    int selectLast;
    Tk_Item *anchorItemPtr;
    int selectAnchor;
    Tk_3DBorder insertBorder;
    insertWidth;
    int insertWidth;
    Tk_Item *focusItemPtr;
    int gotFocus;
    int cursorOn;
} Tk_CanvasTextInfo;

The selBorder field identifies a Tk_3DBorder that should be used for drawing the background under selected
text. selBorderWidth gives the width of the raised border around selected text, in pixels. selFgColorPtr points
to an XColor that describes the foreground color to be used when drawing selected text. selItemPtr points
to the item that is currently selected, or NULL if there is no item selected or if the canvas doesn’t have the
selection. selectFirst and selectLast give the indices of the first and last selected characters in selItemPtr, as
returned by the indexProc for that item. anchorItemPtr points to the item that currently has the selection anchor; this is not necessarily the same as selItemPtr. selectAnchor is an index that identifies the anchor position within anchorItemPtr. insertBorder contains a Tk_3DBorder to use when drawing the insertion cursor; insertWidth gives the total width of the insertion cursor in pixels, and insertBorderWidth gives the width of the raised border around the insertion cursor. focusItemPtr identifies the item that currently has the input focus, or NULL if there is no such item. gotFocus is 1 if the canvas widget has the input focus and 0 otherwise. cursorOn is 1 if the insertion cursor should be drawn in focusItemPtr and 0 if it should not be drawn; this field is toggled on and off by Tk to make the cursor blink.

The structure returned by Tk_CanvasGetTextInfo is shared between Tk and the type managers; typically the type manager calls Tk_CanvasGetTextInfo once when an item is created and then saves the pointer in the item's record. Tk will update information in the Tk_CanvasTextInfo; for example, a configure widget command might change the selBorder field, or a select widget command might change the selectFirst field, or Tk might change cursorOn in order to make the insertion cursor flash on and off during successive redisplays.

Type managers should treat all of the fields of the Tk_CanvasTextInfo structure as read−only, except for selItemPtr, selectFirst, selectLast, and selectAnchor. Type managers may change selectFirst, selectLast, and selectAnchor to adjust for insertions and deletions in the item (but only if the item is the current owner of the selection or anchor, as determined by selItemPtr or anchorItemPtr). If all of the selected text in the item is deleted, the item should set selItemPtr to NULL to indicate that there is no longer a selection.

**KEYWORDS**

canvas, focus, insertion cursor, selection, selection anchor, text

*Copyright ' 1994−1996 Sun Microsystems, Inc.*  
*Copyright ' 1995−1997 Roger E. Critchlow Jr.*
CanvTkwin

NAME

Tk_CanvasTkwin, Tk_CanvasGetCoord, Tk_CanvasDrawableCoords, Tk_CanvasSetStippleOrigin,
Tk_CanvasWindowCoords, Tk_CanvasEventuallyRedraw, Tk_CanvasTagsOption – utility procedures for
canvas type managers

SYNOPSIS

#include <tk.h>
Tk_Window
Tk_CanvasTkwin(canvas)
int
Tk_CanvasGetCoord(interp, canvas, string, doublePtr)
Tk_CanvasDrawableCoords(canvas, x, y, drawableXPtr, drawableYPtr)
Tk_CanvasSetStippleOrigin(canvas, gc)
Tk_CanvasWindowCoords(canvas, x, y, screenXPtr, screenYPtr)
Tk_CanvasEventuallyRedraw(canvas, x1, y1, x2, y2)
Tk_OptionParseProc *Tk_CanvasTagsParseProc;
Tk_OptionPrintProc *Tk_CanvasTagsPrintProc;

ARGUMENTS

Tk_Canvas canvas (in)
A token that identifies a canvas widget.
Tcl_Interp *interp (in/out)
Interpreter to use for error reporting.
CONST char *string (in)
Textual description of a canvas coordinate.
double *doublePtr (out)
Points to place to store a converted coordinate.
double x (in)
An x coordinate in the space of the canvas.
double y (in)
A y coordinate in the space of the canvas.
short *drawableXPtr (out)
Pointer to a location in which to store an x coordinate in the space of the drawable currently being
used to redisplay the canvas.
short *drawableYPtr (out)
Pointer to a location in which to store a y coordinate in the space of the drawable currently being used
to redisplay the canvas.
GC gc (out)
Graphics context to modify.
short *screenXPtr (out)
    Points to a location in which to store the screen coordinate in the canvas window that corresponds to x.
short *screenYPtr (out)
    Points to a location in which to store the screen coordinate in the canvas window that corresponds to y.
int x1 (in)
    Left edge of the region that needs redisplay. Only pixels at or to the right of this coordinate need to be redisplayed.
int y1 (in)
    Top edge of the region that needs redisplay. Only pixels at or below this coordinate need to be redisplayed.
int x2 (in)
    Right edge of the region that needs redisplay. Only pixels to the left of this coordinate need to be redisplayed.
int y2 (in)
    Bottom edge of the region that needs redisplay. Only pixels above this coordinate need to be redisplayed.

DESCRIPTION

These procedures are called by canvas type managers to perform various utility functions.

Tk_CanvasTkwin returns the Tk_Window associated with a particular canvas.

Tk_CanvasGetCoord translates a string specification of a coordinate (such as 2p or 1.6c) into a double-precision canvas coordinate. If string is a valid coordinate description then Tk_CanvasGetCoord stores the corresponding canvas coordinate at *doublePtr and returns TCL_OK. Otherwise it stores an error message in interp-&gt;result and returns TCL_ERROR.

Tk_CanvasDrawableCoords is called by type managers during redisplay to compute where to draw things. Given x and y coordinates in the space of the canvas, Tk_CanvasDrawableCoords computes the corresponding pixel in the drawable that is currently being used for redisplay; it returns those coordinates in *drawableXPtr and *drawableYPtr. This procedure should not be invoked except during redisplay.

Tk_CanvasSetStippleOrigin is also used during redisplay. It sets the stipple origin in gc so that stipple drawings with gc in the current offscreen pixmap will line up with stipple drawings with origin (0,0) in the canvas's actual window. Tk_CanvasSetStippleOrigin is needed in order to guarantee that stipple patterns line up properly when the canvas is redisplayed in small pieces. Redisplays are carried out in double-buffered fashion where a piece of the canvas is redrawn in an offscreen pixmap and then copied back onto the screen. In this approach the stipple origins in graphics contexts need to be adjusted during each redisplay to compensate for the position of the off-screen pixmap relative to the window. If an item is being drawn with stiples, its type manager typically calls Tk_CanvasSetStippleOrigin just before using gc to draw something; after it is finished drawing, the type manager calls XSetTSOrigin to restore the origin in gc back to (0,0) (the restore is needed because graphics contexts are shared, so they cannot be modified permanently).
Tk_CanvasWindowCoords is similar to Tk_CanvasDrawableCoords except that it returns coordinates in the canvas's window on the screen, instead of coordinates in an off−screen pixmap.

Tk_CanvasEventuallyRedraw may be invoked by a type manager to inform Tk that a portion of a canvas needs to be redrawn. The x1, y1, x2, and y2 arguments specify the region that needs to be redrawn, in canvas coordinates. Type managers rarely need to invoke Tk_CanvasEventuallyRedraw, since Tk can normally figure out when an item has changed and make the redisplay request on its behalf (this happens, for example whenever Tk calls a configureProc or scaleProc). The only time that a type manager needs to call Tk_CanvasEventuallyRedraw is if an item has changed on its own without being invoked through one of the procedures in its Tk_ItemType; this could happen, for example, in an image item if the image is modified using image commands.

Tk_CanvasTagsParseProc and Tk_CanvasTagsPrintProc are procedures that handle the −tags option for canvas items. The code of a canvas type manager won't call these procedures directly, but will use their addresses to create a Tk_CustomOption structure for the −tags option. The code typically looks like this:

```
static Tk_CustomOption tagsOption = {Tk_CanvasTagsParseProc,
   Tk_CanvasTagsPrintProc, (ClientData) NULL);

static Tk_ConfigSpec configSpecs[] = {
   ...  
   {TK_CONFIG_CUSTOM, "-tags", (char *) NULL, (char *) NULL,
    (char *) NULL, 0, TK_CONFIG_NULL_OK, &tagsOption},
   ...  
};
```

KEYWORDS

canvas, focus, item type, redisplay, selection, type manager

Copyright ' 1994−1996 Sun Microsystems, Inc.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
ClrSelect

NAME

Tk_ClearSelection – Deselect a selection

SYNOPSIS

#include <tk.h>
Tk_ClearSelection(tkwin, selection)

ARGUMENTS

Tk_Window tkwin (in)
The selection will be cleared from the display containing this window.
Atom selection (in)
The name of selection to be cleared.

DESCRIPTION

Tk_ClearSelection cancels the selection specified by the atom selection for the display containing tkwin. The selection need not be in tkwin itself or even in tkwin's application. If there is a window anywhere on tkwin's display that owns selection, the window will be notified and the selection will be cleared. If there is no owner for selection on the display, then the procedure has no effect.

KEYWORDS

clear, selection

Copyright ' 1992−1994 The Regents of the University of California.
Copyright ' 1994−1996 Sun Microsystems, Inc.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
Clipboard

NAME

Tk_ClipboardClear, Tk_ClipboardAppend – Manage the clipboard

SYNOPSIS

#include <tk.h>

int Tk_ClipboardClear(interp, tkwin)
int Tk_ClipboardAppend(interp, tkwin, target, format, buffer)

ARGUMENTS

Tcl_Interp *interp (in)
Interpreter to use for reporting errors.
Tk_Window tkwin (in)
Window that determines which display's clipboard to manipulate.
Atom target (in)
Conversion type for this clipboard item; has same meaning as target argument to Tk_CreateSelHandler.
Atom format (in)
Representation to use when data is retrieved; has same meaning as format argument to Tk_CreateSelHandler.
char *buffer (in)
Null terminated string containing the data to be appended to the clipboard.

DESCRIPTION

These two procedures manage the clipboard for Tk. The clipboard is typically managed by calling Tk_ClipboardClear once, then calling Tk_ClipboardAppend to add data for any number of targets.

Tk_ClipboardClear claims the CLIPBOARD selection and frees any data items previously stored on the clipboard in this application. It normally returns TCL_OK, but if an error occurs it returns TCL_ERROR and leaves an error message in interp−>result. Tk_ClipboardClear must be called before a sequence of Tk_ClipboardAppend calls can be issued.

Tk_ClipboardAppend appends a buffer of data to the clipboard. The first buffer for a given target determines the format for that target. Any successive appends for that target must have the same format or an error will be returned. Tk_ClipboardAppend returns TCL_OK if the buffer is successfully copied onto the clipboard. If the clipboard is not currently owned by the application, either because Tk_ClipboardClear has not been called or because ownership of the clipboard has changed since the last call to Tk_ClipboardClear,
Tk_ClipboardAppend returns TCL_ERROR and leaves an error message in interp->result.

In order to guarantee atomicity, no event handling should occur between Tk_ClipboardClear and the following Tk_ClipboardAppend calls (otherwise someone could retrieve a partially completed clipboard or claim ownership away from this application).

Tk_ClipboardClear may invoke callbacks, including arbitrary Tcl scripts, as a result of losing the CLIPBOARD selection, so any calling function should take care to be reentrant at the point Tk_ClipboardClear is invoked.

KEYWORDS
append, clipboard, clear, format, type

Copyright ' 1994 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
TextLayout

NAME

Tk_ComputeTextLayout, Tk_FreeTextLayout, Tk_DrawTextLayout, Tk_UnderlineTextLayout,
Tk_PointToChar, Tk_CharBbox, Tk_DistanceToTextLayout, Tk_IntersectTextLayout,
Tk_TextLayoutToPostscript – routines to measure and display single−font, multi−line, justified text.

SYNOPSIS

#include <tk.h>
Tk_TextLayout
Tk_ComputeTextLayout(tkfont, string, numChars, wrapLength, justify, flags, widthPtr, heightPtr)
void
Tk_FreeTextLayout(layout)
void
Tk_DrawTextLayout(display, drawable, gc, layout, x, y, firstChar, lastChar)
void
Tk_UnderlineTextLayout(display, drawable, gc, layout, x, y, underline)
int
Tk_PointToChar(layout, x, y)
int
Tk_CharBbox(layout, index, xPtr, yPtr, widthPtr, heightPtr)
int
Tk_DistanceToTextLayout(layout, x, y)
int
Tk_IntersectTextLayout(layout, x, y, width, height)
void
Tk_TextLayoutToPostscript(interp, layout)

ARGUMENTS

Tk_Font tkfont (in)
Font to use when constructing and displaying a text layout. The tkfont must remain valid for the
lifetime of the text layout. Must have been returned by a previous call to Tk_GetFont.
const char *string (in)
Potentially multi−line string whose dimensions are to be computed and stored in the text layout. The
string must remain valid for the lifetime of the text layout.
int numChars (in)
The number of characters to consider from string. If numChars is less than 0, then assumes string is
null terminated and uses Tcl_NumUtfChars to determine the length of string.
int wrapLength (in)
Longest permissible line length, in pixels. Lines in string will automatically be broken at word
boundaries and wrapped when they reach this length. If \textit{wrapLength} is too small for even a single character to fit on a line, it will be expanded to allow one character to fit on each line. If \textit{wrapLength} is $\leq 0$, there is no automatic wrapping; lines will get as long as they need to be and only wrap if a newline/return character is encountered.

\textbf{Tk\_Justify justify (in)}

How to justify the lines in a multi-line text layout. Possible values are TK\_JUSTIFY\_LEFT, TK\_JUSTIFY\_CENTER, or TK\_JUSTIFY\_RIGHT. If the text layout only occupies a single line, then justify is irrelevant.

\textbf{int flags (in)}

Various flag bits OR-ed together. TK\_IGNORE\_TABS means that tab characters should not be expanded to the next tab stop. TK\_IGNORE\_NEWLINES means that newline/return characters should not cause a line break. If either tabs or newlines/returns are ignored, then they will be treated as regular characters, being measured and displayed in a platform-dependent manner as described in \textbf{Tk\_MeasureChars}, and will not have any special behaviors.

\textbf{int *widthPtr (out)}

If non-NULL, filled with either the width, in pixels, of the widest line in the text layout, or the width, in pixels, of the bounding box for the character specified by \textit{index}.

\textbf{int *heightPtr (out)}

If non-NULL, filled with either the total height, in pixels, of all the lines in the text layout, or the height, in pixels, of the bounding box for the character specified by \textit{index}.

\textbf{Tk\_TextLayout layout (in)}

A token that represents the cached layout information about the single-font, multi-line, justified piece of text. This token is returned by \textbf{Tk\_ComputeTextLayout}.

\textbf{Display *display (in)}

Display on which to draw.

\textbf{Drawable drawable (in)}

Window or pixmap in which to draw.

\textbf{GC gc (in)}

Graphics context to use for drawing text layout. The font selected in this GC must correspond to the \textit{tkfont} used when constructing the text layout.

\textbf{int x, y (in)}

Point, in pixels, at which to place the upper-left hand corner of the text layout when it is being drawn, or the coordinates of a point (with respect to the upper-left hand corner of the text layout) to check against the text layout.

\textbf{int firstChar (in)}

The index of the first character to draw from the given text layout. The number 0 means to draw from the beginning.

\textbf{int lastChar (in)}

The index of the last character up to which to draw. The character specified by \textit{lastChar} itself will not be drawn. A number less than 0 means to draw all characters in the text layout.

\textbf{int underline (in)}

Index of the single character to underline in the text layout, or a number less than 0 for no underline.

\textbf{int index (in)}

The index of the character whose bounding box is desired. The bounding box is computed with respect to the upper-left hand corner of the text layout.
int *xPtr, *yPtr (out)
Filled with the upper−left hand corner, in pixels, of the bounding box for the character specified by index. Either or both xPtr and yPtr may be NULL, in which case the corresponding value is not calculated.

int width, height (in)
Specifies the width and height, in pixels, of the rectangular area to compare for intersection against the text layout.

Tcl_Interp *interp (out)
Postscript code that will print the text layout is appended to interp−>result.

DESCRIPTION

These routines are for measuring and displaying single−font, multi−line, justified text. To measure and display simple single−font, single−line strings, refer to the documentation for Tk_MeasureChars. There is no programming interface in the core of Tk that supports multi−font, multi−line text; support for that behavior must be built on top of simpler layers. Note that unlike the lower level text display routines, the functions described here all operate on character−oriented lengths and indices rather than byte−oriented values. See the description of Tcl_UtfAtIndex for more details on converting between character and byte offsets.

The routines described here are built on top of the programming interface described in the Tk_MeasureChars documentation. Tab characters and newline/return characters may be treated specially by these procedures, but all other characters are passed through to the lower level.

Tk_ComputeTextLayout computes the layout information needed to display a single−font, multi−line, justified string of text and returns a Tk_TextLayout token that holds this information. This token is used in subsequent calls to procedures such as Tk_DrawTextLayout, Tk_DistanceToTextLayout, and Tk_FreeTextLayout. The string and tkfont used when computing the layout must remain valid for the lifetime of this token.

Tk_FreeTextLayout is called to release the storage associated with layout when it is no longer needed. A layout should not be used in any other text layout procedures once it has been released.

Tk_DrawTextLayout uses the information in layout to display a single−font, multi−line, justified string of text at the specified location.

Tk_UnderlineTextLayout uses the information in layout to display an underline below an individual character. This procedure does not draw the text, just the underline. To produce natively underlined text, an underlined font should be constructed and used. All characters, including tabs, newline/return characters, and spaces at the ends of lines, can be underlined using this method. However, the underline will never be drawn outside of the computed width of layout; the underline will stop at the edge for any character that would extend partially outside of layout, and the underline will not be visible at all for any character that would be located completely outside of the layout.

Tk_PointToChar uses the information in layout to determine the character closest to the given point. The point is specified with respect to the upper−left hand corner of the layout, which is considered to be located at
Any point whose y−value is less than 0 will be considered closest to the first character in the text layout; any point whose y−value is greater than the height of the text layout will be considered closest to the last character in the text layout. Any point whose x−value is less than 0 will be considered closest to the first character on that line; any point whose x−value is greater than the width of the text layout will be considered closest to the last character on that line. The return value is the index of the character that was closest to the point. Given a layout with no characters, the value 0 will always be returned, referring to a hypothetical zero−width placeholder character.

Tk_CharBbox uses the information in layout to return the bounding box for the character specified by index. The width of the bounding box is the advance width of the character, and does not include any left or right bearing. Any character that extends partially outside of layout is considered to be truncated at the edge. Any character that would be located completely outside of layout is considered to be zero−width and pegged against the edge. The height of the bounding box is the line height for this font, extending from the top of the ascent to the bottom of the descent; information about the actual height of individual letters is not available. For measurement purposes, a layout that contains no characters is considered to contain a single zero−width placeholder character at index 0. If index was not a valid character index, the return value is 0 and *xPtr, *yPtr, *widthPtr, and *heightPtr are unmodified. Otherwise, if index did specify a valid, the return value is non−zero, and *xPtr, *yPtr, *widthPtr, and *heightPtr are filled with the bounding box information for the character. If any of xPtr, yPtr, widthPtr, or heightPtr are NULL, the corresponding value is not calculated or stored.

Tk_DistanceToTextLayout computes the shortest distance in pixels from the given point (x, y) to the characters in layout. Newline/return characters and non−displaying space characters that occur at the end of individual lines in the layout are ignored for hit detection purposes, but tab characters are not. The return value is 0 if the point actually hits the layout. If the point didn't hit the layout then the return value is the distance in pixels from the point to the layout.

Tk_IntersectTextLayout determines whether a layout lies entirely inside, entirely outside, or overlaps a given rectangle. Newline/return characters and non−displaying space characters that occur at the end of individual lines in the layout are ignored for intersection calculations. The return value is −1 if the layout is entirely outside of the rectangle, 0 if it overlaps, and 1 if it is entirely inside of the rectangle.

Tk_TextLayoutToPostscript outputs code consisting of a Postscript array of strings that represent the individual lines in layout. It is the responsibility of the caller to take the Postscript array of strings and add some Postscript function operate on the array to render each of the lines. The code that represents the Postscript array of strings is appended to interp−>result.

DISPLAY MODEL

When measuring a text layout, space characters that occur at the end of a line are ignored. The space characters still exist and the insertion point can be positioned amongst them, but their additional width is ignored when justifying lines or returning the total width of a text layout. All end−of−line space characters are considered to be attached to the right edge of the line; this behavior is logical for left−justified text and reasonable for center−justified text, but not very useful when editing right−justified text. Spaces are considered variable width characters; the first space that extends past the edge of the text layout is clipped to
the edge, and any subsequent spaces on the line are considered zero width and pegged against the edge. Space characters that occur in the middle of a line of text are not suppressed and occupy their normal space width.

Tab characters are not ignored for measurement calculations. If wrapping is turned on and there are enough tabs on a line, the next tab will wrap to the beginning of the next line. There are some possible strange interactions between tabs and justification; tab positions are calculated and the line length computed in a left-justified world, and then the whole resulting line is shifted so it is centered or right-justified, causing the tab columns not to align any more.

When wrapping is turned on, lines may wrap at word breaks (space or tab characters) or newline/returns. A dash or hyphen character in the middle of a word is not considered a word break. **Tk_ComputeTextLayout** always attempts to place at least one word on each line. If it cannot because the \textit{wrapLength} is too small, the word will be broken and as much as fits placed on the line and the rest on subsequent line(s). If \textit{wrapLength} is so small that not even one character can fit on a given line, the \textit{wrapLength} is ignored for that line and one character will be placed on the line anyhow. When wrapping is turned off, only newline/return characters may cause a line break.

When a text layout has been created using an underlined \textit{tkfont}, then any space characters that occur at the end of individual lines, newlines/returns, and tabs will not be displayed underlined when **Tk_DrawTextLayout** is called, because those characters are never actually drawn – they are merely placeholders maintained in the layout.

**KEYWORDS**

\textit{font}
ConfigWidg

NAME

Tk_ConfigureWidget, Tk_ConfigureInfo, Tk_ConfigureValue, Tk_FreeOptions – process configuration options for widgets

SYNOPSIS

ARGUMENTS

DESCRIPTION

TK CONFIG ACTIVE_CURSOR
TK CONFIG ANCHOR
TK CONFIG BITMAP
TK CONFIG BOOLEAN
TK CONFIG BORDER
TK CONFIG CAP_STYLE
TK CONFIG COLOR
TK CONFIG CURSOR
TK CONFIG CUSTOM
TK CONFIG DOUBLE
TK CONFIG END
TK CONFIG FONT
TK CONFIG INT
TK CONFIG JOIN_STYLE
TK CONFIG JUSTIFY
TK CONFIG MM
TK CONFIG PIXELS
TK CONFIG RELIEF
TK CONFIG STRING
TK CONFIG SYNONYM
TK CONFIG UID
TK CONFIG WINDOW

GROUPED ENTRIES

FLAGS

TK CONFIG COLOR ONLY
TK CONFIG MONO ONLY
TK CONFIG NULL OK
TK CONFIG DONT SET DEFAULT
TK CONFIG OPTION SPECIFIED

TK OFFSET
TK CONFIGUREINFO
TK CONFIGUREVALUE
TK FREEOPTIONS
CUSTOM OPTION TYPES
EXAMPLES
NAME

Tk_ConfigureWidget, Tk_ConfigureInfo, Tk_ConfigureValue, Tk_FreeOptions – process configuration options for widgets

SYNOPSIS

#include <tk.h>
int
Tk_ConfigureWidget(interp, tkwin, specs, argc, argv, widgRec, flags)
int
Tk_ConfigureInfo(interp, tkwin, specs, widgRec, argvName, flags)
int
Tk_ConfigureValue(interp, tkwin, specs, widgRec, argvName, flags)
Tk_FreeOptions(specs, widgRec, display, flags)

ARGUMENTS

Tcl_Interp *interp (in)
    Interpreter to use for returning error messages.
Tk_Window tkwin (in)
    Window used to represent widget (needed to set up X resources).
Tk_ConfigSpec *specs (in)
    Pointer to table specifying legal configuration options for this widget.
int argc (in)
    Number of arguments in argv.
CONST char **argv (in)
    Command-line options for configuring widget.
char *widgRec (in/out)
    Points to widget record structure. Fields in this structure get modified by Tk_ConfigureWidget to hold configuration information.
int flags (in)
    If non-zero, then it specifies an OR-ed combination of flags that control the processing of configuration information. TK_CONFIG_ARGV_ONLY causes the option database and defaults to be ignored, and flag bits TK_CONFIG_USER_BIT and higher are used to selectively disable entries in specs.
type name type (in)
    The name of the type of a widget record.
field name field (in)
    The name of a field in records of type type.
CONST char *argvName (in)
The name used on Tcl command lines to refer to a particular option (e.g. when creating a widget or
invoking the configure widget command). If non–NULL, then information is returned only for this
option. If NULL, then information is returned for all available options.

Display*display (in)
Display containing widget whose record is being freed; needed in order to free up resources.

DESCRIPTION

Note: Tk_ConfigureWidget should be replaced with the new Tcl_Obj based API Tk_SetOptions. The old
interface is retained for backward compatibility.

Tk_ConfigureWidget is called to configure various aspects of a widget, such as colors, fonts, border width,
etc. It is intended as a convenience procedure to reduce the amount of code that must be written in individual
widget managers to handle configuration information. It is typically invoked when widgets are created, and
again when the configure command is invoked for a widget. Although intended primarily for widgets,
Tk_ConfigureWidget can be used in other situations where argc–argv information is to be used to fill in a
record structure, such as configuring graphical elements for a canvas widget or entries of a menu.

Tk_ConfigureWidget processes a table specifying the configuration options that are supported (specs) and a
collection of command–line arguments (argc and argv) to fill in fields of a record (widgRec). It uses the
option database and defaults specified in specs to fill in fields of widgRec that are not specified in argv.
Tk_ConfigureWidget normally returns the value TCL_OK; in this case it does not modify interp. If an error
occurs then TCL_ERROR is returned and Tk_ConfigureWidget will leave an error message in
interp–>result in the standard Tcl fashion. In the event of an error return, some of the fields of widgRec could
already have been set, if configuration information for them was successfully processed before the error
occurred. The other fields will be set to reasonable initial values so that Tk_FreeOptions can be called for
cleanup.

The specs array specifies the kinds of configuration options expected by the widget. Each of its entries
specifies one configuration option and has the following structure:

typedef struct {
    type int
    argvName;
    dbName; *
    dbClass;
    defValue;
    offset;
    specFlags;
    Tk_CustomOption *customPtr;
} Tk_ConfigSpec;

The type field indicates what type of configuration option this is (e.g. TK_CONFIG_COLOR for a color
value, or TK_CONFIG_INT for an integer value). The type field indicates how to use the value of the option
(more on this below). The argvName field is a string such as ``–font'' or ``–bg'', which is compared with the
values in argv (if argvName is NULL it means this is a grouped entry; see GROUPED ENTRIES below). The
dbName and dbClass fields are used to look up a value for this option in the option database. The defValue
field specifies a default value for this configuration option if no value is specified in either argv or the option database. Offset indicates where in widgRec to store information about this option, and specFlags contains additional information to control the processing of this configuration option (see FLAGS below). The last field, customPtr, is only used if type is TK_CONFIG_CUSTOM; see CUSTOM OPTION TYPES below.

Tk_ConfigureWidget first processes argv to see which (if any) configuration options are specified there. argv must contain an even number of fields; the first of each pair of fields must match the argvName of some entry in specs (unique abbreviations are acceptable), and the second field of the pair contains the value for that configuration option. If there are entries in spec for which there were no matching entries in argv, Tk_ConfigureWidget uses the dbName and dbClass fields of the specs entry to probe the option database; if a value is found, then it is used as the value for the option. Finally, if no entry is found in the option database, the defValue field of the specs entry is used as the value for the configuration option. If the defValue is NULL, or if the TK_CONFIG_DONT_SET_DEFAULT bit is set in flags, then there is no default value and this specs entry will be ignored if no value is specified in argv or the option database.

Once a string value has been determined for a configuration option, Tk_ConfigureWidget translates the string value into a more useful form, such as a color if type is TK_CONFIG_COLOR or an integer if type is TK_CONFIG_INT. This value is then stored in the record pointed to by widgRec. This record is assumed to contain information relevant to the manager of the widget; its exact type is unknown to Tk_ConfigureWidget. The offset field of each specs entry indicates where in widgRec to store the information about this configuration option. You should use the Tk_Offset macro to generate offset values (see below for a description of Tk_Offset). The location indicated by widgRec and offset will be referred to as the "target" in the descriptions below.

The type field of each entry in specs determines what to do with the string value of that configuration option. The legal values for type, and the corresponding actions, are:

**TK_CONFIG_ACTIVE_CURSOR**

The value must be an ASCII string identifying a cursor in a form suitable for passing to Tk_GetCursor. The value is converted to a Tk_Cursor by calling Tk_GetCursor and the result is stored in the target. In addition, the resulting cursor is made the active cursor for tkwin by calling XDefineCursor. If TK_CONFIG_NULL_OK is specified in specFlags then the value may be an empty string, in which case the target and tkwin’s active cursor will be set to None. If the previous value of the target wasn’t None, then it is freed by passing it to Tk_FreeCursor.

**TK_CONFIG_ANCHOR**

The value must be an ASCII string identifying an anchor point in one of the ways accepted by Tk_GetAnchor. The string is converted to a Tk_Anchor by calling Tk_GetAnchor and the result is stored in the target.

**TK_CONFIG_BITMAP**

The value must be an ASCII string identifying a bitmap in a form suitable for passing to Tk_GetBitmap. The value is converted to a Pixmap by calling Tk_GetBitmap and the result is stored in the target. If TK_CONFIG_NULL_OK is specified in specFlags then the value may be an empty string, in which case the target and tkwin's active cursor will be set to None. If the previous value of the target wasn't None, then it is freed by passing it to Tk_FreeBitmap.

**TK_CONFIG_BOOLEAN**
The value must be an ASCII string specifying a boolean value. Any of the values `true', `yes', `on', or `1', or an abbreviation of one of these values, means true; any of the values `false', `no', `off', or `0', or an abbreviation of one of these values, means false. The target is expected to be an integer; for true values it will be set to 1 and for false values it will be set to 0.

**TK_CONFIG_BORDER**
The value must be an ASCII string identifying a border color in a form suitable for passing to `Tk_Get3DBorder'. The value is converted to a (`Tk_3DBorder *`) by calling `Tk_Get3DBorder` and the result is stored in the target. If TK_CONFIG_NULL_OK is specified in `specFlags` then the value may be an empty string, in which case the target will be set to NULL. If the previous value of the target wasn't NULL, then it is freed by passing it to `Tk_Free3DBorder`.

**TK_CONFIG_CAP_STYLE**
The value must be an ASCII string identifying a cap style in one of the ways accepted by `Tk_GetCapStyle`. The string is converted to an integer value corresponding to the cap style by calling `Tk_GetCapStyle` and the result is stored in the target.

**TK_CONFIG_COLOR**
The value must be an ASCII string identifying a color in a form suitable for passing to `Tk_GetColor`. The value is converted to an (`XColor *`) by calling `Tk_GetColor` and the result is stored in the target. If TK_CONFIG_NULL_OK is specified in `specFlags` then the value may be an empty string, in which case the target will be set to `None`. If the previous value of the target wasn't NULL, then it is freed by passing it to `Tk_FreeColor`.

**TK_CONFIG_CURSOR**
This option is identical to **TK_CONFIG_ACTIVE_CURSOR** except that the new cursor is not made the active one for `tkwin`.

**TK_CONFIG_CUSTOM**
This option allows applications to define new option types. The `customPtr` field of the entry points to a structure defining the new option type. See the section CUSTOM OPTION TYPES below for details.

**TK_CONFIG_DOUBLE**
The value must be an ASCII floating−point number in the format accepted by `strtol`. The string is converted to a `double` value, and the value is stored in the target.

**TK_CONFIG_END**
Marks the end of the table. The last entry in `specs` must have this type; all of its other fields are ignored and it will never match any arguments.

**TK_CONFIG_FONT**
The value must be an ASCII string identifying a font in a form suitable for passing to `Tk_GetFont`. The value is converted to a `Tk_Font` by calling `Tk_GetFont` and the result is stored in the target. If TK_CONFIG_NULL_OK is specified in `specFlags` then the value may be an empty string, in which case the target will be set to NULL. If the previous value of the target wasn't NULL, then it is freed by passing it to `Tk_FreeFont`.

**TK_CONFIG_INT**
The value must be an ASCII integer string in the format accepted by `strtol` (e.g. `"0"` and `"0x"` prefixes may be used to specify octal or hexadecimal numbers, respectively). The string is converted to an integer value and the integer is stored in the target.

**TK_CONFIG_JOIN_STYLE**
The value must be an ASCII string identifying a join style in one of the ways accepted by...
Tk_GetJoinStyle. The string is converted to an integer value corresponding to the join style by calling Tk_GetJoinStyle and the result is stored in the target.

TK_CONFIG_JUSTIFY
The value must be an ASCII string identifying a justification method in one of the ways accepted by Tk_GetJustify. The string is converted to a Tk_Justify by calling Tk_GetJustify and the result is stored in the target.

TK_CONFIG_MM
The value must specify a screen distance in one of the forms acceptable to Tk_GetScreenMM. The string is converted to double-precision floating-point distance in millimeters and the value is stored in the target.

TK_CONFIG_PIXELS
The value must specify screen units in one of the forms acceptable to Tk_GetPixels. The string is converted to an integer distance in pixels and the value is stored in the target.

TK_CONFIG_RELIEF
The value must be an ASCII string identifying a relief in a form suitable for passing to Tk_GetRelief. The value is converted to an integer relief value by calling Tk_GetRelief and the result is stored in the target.

TK_CONFIG_STRING
A copy of the value is made by allocating memory space with malloc and copying the value into the dynamically-allocated space. A pointer to the new string is stored in the target. If TK_CONFIG_NULL_OK is specified in specFlags then the value may be an empty string, in which case the target will be set to NULL. If the previous value of the target wasn't NULL, then it is freed by passing it to free.

TK_CONFIG_SYNONYM
This type value identifies special entries in specs that are synonyms for other entries. If an argv value matches the argvName of a TK_CONFIG_SYNONYM entry, the entry isn't used directly. Instead, Tk_ConfigureWidget searches specs for another entry whose argvName is the same as the dbName field in the TK_CONFIG_SYNONYM entry; this new entry is used just as if its argvName had matched the argv value. The synonym mechanism allows multiple argv values to be used for a single configuration option, such as ``-background'' and ``-bg''.

TK_CONFIG_UID
The value is translated to a Tk_Uid (by passing it to Tk_GetUid). The resulting value is stored in the target. If TK_CONFIG_NULL_OK is specified in specFlags and the value is an empty string then the target will be set to NULL.

TK_CONFIG_WINDOW
The value must be a window path name. It is translated to a Tk_Window token and the token is stored in the target.

GROUPED ENTRIES

In some cases it is useful to generate multiple resources from a single configuration value. For example, a color name might be used both to generate the background color for a widget (using TK_CONFIG_COLOR) and to generate a 3-D border to draw around the widget (using TK_CONFIG_BORDER). In cases like this it is possible to specify that several consecutive entries in specs are to be treated as a group. The first entry is used to determine a value (using its argvName, dbName, dbClass, and defValue fields). The value will be
processed several times (one for each entry in the group), generating multiple different resources and modifying multiple targets within *widgRec*. Each of the entries after the first must have a NULL value in its *argvName* field; this indicates that the entry is to be grouped with the entry that precedes it. Only the *type* and *offset* fields are used from these follow–on entries.

**FLAGS**

The *flags* argument passed to *Tk_ConfigureWidget* is used in conjunction with the *specFlags* fields in the entries of *specs* to provide additional control over the processing of configuration options. These values are used in three different ways as described below.

First, if the *flags* argument to *Tk_ConfigureWidget* has the TK_CONFIG_ARGV_ONLY bit set (i.e., *flags* | TK_CONFIG_ARGV_ONLY != 0), then the option database and *defValue* fields are not used. In this case, if an entry in *specs* doesn't match a field in *argv* then nothing happens: the corresponding target isn't modified. This feature is useful when the goal is to modify certain configuration options while leaving others in their current state, such as when a *configure* widget command is being processed.

Second, the *specFlags* field of an entry in *specs* may be used to control the processing of that entry. Each *specFlags* field may consists of an OR–ed combination of the following values:

- **TK_CONFIG_COLOR_ONLY**
  If this bit is set then the entry will only be considered if the display for *tkwin* has more than one bit plane. If the display is monochromatic then this *specs* entry will be ignored.

- **TK_CONFIG_MONO_ONLY**
  If this bit is set then the entry will only be considered if the display for *tkwin* has exactly one bit plane. If the display is not monochromatic then this *specs* entry will be ignored.

- **TK_CONFIG_NULL_OK**
  This bit is only relevant for some types of entries (see the descriptions of the various entry types above). If this bit is set, it indicates that an empty string value for the field is acceptable and if it occurs then the target should be set to NULL or *None*, depending on the type of the target. This flag is typically used to allow a feature to be turned off entirely, e.g. set a cursor value to *None* so that a window simply inherits its parent's cursor. If this bit isn't set then empty strings are processed as strings, which generally results in an error.

- **TK_CONFIG_DONT_SET_DEFAULT**
  If this bit is one, it means that the *defValue* field of the entry should only be used for returning the default value in *Tk_ConfigureInfo*. In calls to *Tk_ConfigureWidget* no default will be supplied for entries with this flag set; it is assumed that the caller has already supplied a default value in the target location. This flag provides a performance optimization where it is expensive to process the default string: the client can compute the default once, save the value, and provide it before calling *Tk_ConfigureWidget*.

- **TK_CONFIG_OPTION_SPECIFIED**
  This bit is set and cleared by *Tk_ConfigureWidget*. Whenever *Tk_ConfigureWidget* returns, this bit will be set in all the entries where a value was specified in *argv*. It will be zero in all other entries. This bit provides a way for clients to determine which values actually changed in a call to *Tk_ConfigureWidget*. 
The TK_CONFIG_MONO_ONLY and TK_CONFIG_COLOR_ONLY flags are typically used to specify different default values for monochrome and color displays. This is done by creating two entries in specs that are identical except for their defValue and specFlags fields. One entry should have the value TK_CONFIG_MONO_ONLY in its specFlags and the default value for monochrome displays in its defValue; the other entry entry should have the value TK_CONFIG_COLOR_ONLY in its specFlags and the appropriate defValue for color displays.

Third, it is possible to use flags and specFlags together to selectively disable some entries. This feature is not needed very often. It is useful in cases where several similar kinds of widgets are implemented in one place. It allows a single specs table to be created with all the configuration options for all the widget types. When processing a particular widget type, only entries relevant to that type will be used. This effect is achieved by setting the high−order bits (those in positions equal to or greater than TK_CONFIG_USER_BIT) in specFlags values or in flags. In order for a particular entry in specs to be used, its high−order bits must match exactly the high−order bits of the flags value passed to Tk_ConfigureWidget. If a specs table is being used for N different widget types, then N of the high−order bits will be used. Each specs entry will have one of more of those bits set in its specFlags field to indicate the widget types for which this entry is valid. When calling Tk_ConfigureWidget, flags will have a single one of these bits set to select the entries for the desired widget type. For a working example of this feature, see the code in tkButton.c.

**TK_OFFSET**

The Tk_Offset macro is provided as a safe way of generating the offset values for entries in Tk_ConfigSpec structures. It takes two arguments: the name of a type of record, and the name of a field in that record. It returns the byte offset of the named field in records of the given type.

**TK_CONFIGUREINFO**

The Tk_ConfigureInfo procedure may be used to obtain information about one or all of the options for a given widget. Given a token for a window (tkwin), a table describing the configuration options for a class of widgets (specs), a pointer to a widget record containing the current information for a widget (widgRec), and a NULL argvName argument, Tk_ConfigureInfo generates a string describing all of the configuration options for the window. The string is placed in interp−>result. Under normal circumstances it returns TCL_OK; if an error occurs then it returns TCL_ERROR and interp−>result contains an error message.

If argvName is NULL, then the value left in interp−>result by Tk_ConfigureInfo consists of a list of one or more entries, each of which describes one configuration option (i.e. one entry in specs). Each entry in the list will contain either two or five values. If the corresponding entry in specs has type TK_CONFIG_SYNONYM, then the list will contain two values: the argvName for the entry and the dbName (synonym name). Otherwise the list will contain five values: argvName, dbName, dbClass, defValue, and current value. The current value is computed from the appropriate field of widgRec by calling procedures like Tk_NameOfColor.

If the argvName argument to Tk_ConfigureInfo is non−NULL, then it indicates a single option, and information is returned only for that option. The string placed in interp−>result will be a list containing two or five values as described above; this will be identical to the corresponding sublist that would have been returned if argvName had been NULL.
The flags argument to Tk_ConfigureInfo is used to restrict the specs entries to consider, just as for Tk_ConfigureWidget.

TK_CONFIGUREVALUE

Tk_ConfigureValue takes arguments similar to Tk_ConfigureInfo; instead of returning a list of values, it just returns the current value of the option given by argvName (argvName must not be NULL). The value is returned in interp->result and TCL_OK is normally returned as the procedure's result. If an error occurs in Tk_ConfigureValue (e.g., argvName is not a valid option name), TCL_ERROR is returned and an error message is left in interp->result. This procedure is typically called to implement cget widget commands.

TK_FREEOPTIONS

The Tk_FreeOptions procedure may be invoked during widget cleanup to release all of the resources associated with configuration options. It scans through specs and for each entry corresponding to a resource that must be explicitly freed (e.g., those with type TK_CONFIG_COLOR), it frees the resource in the widget record. If the field in the widget record doesn't refer to a resource (e.g. it contains a null pointer) then no resource is freed for that entry. After freeing a resource, Tk_FreeOptions sets the corresponding field of the widget record to null.

CUSTOM OPTION TYPES

Applications can extend the built-in configuration types with additional configuration types by writing procedures to parse and print options of the a type and creating a structure pointing to those procedures:

typedef struct Tk_CustomOption {
    Tk_OptionParseProc *parseProc;
    Tk_OptionPrintProc *printProc;
    ClientData clientData;
} Tk_CustomOption;

typedef int Tk_OptionParseProc(
    ClientData clientData,
    Tcl_Interp *interp,
    Tk_Window *tkwin,
    char *value,
    char *widgRec,
    int offset);

typedef char *Tk_OptionPrintProc(
    ClientData clientData,
    Tk_Window *tkwin,
    char *widgRec,
    int offset,
    Tcl_FreeProc **freeProcPtr);

The Tk_CustomOption structure contains three fields, which are pointers to the two procedures and a clientData value to be passed to those procedures when they are invoked. The clientData value typically
points to a structure containing information that is needed by the procedures when they are parsing and printing options.

The `parseProc` procedure is invoked by `Tk_ConfigureWidget` to parse a string and store the resulting value in the widget record. The `clientData` argument is a copy of the `clientData` field in the Tk_CustomOption structure. The `interp` argument points to a Tcl interpreter used for error reporting. `Tkwin` is a copy of the `tkwin` argument to `Tk_ConfigureWidget`. The `value` argument is a string describing the value for the option; it could have been specified explicitly in the call to `Tk_ConfigureWidget` or it could come from the option database or a default. `Value` will never be a null pointer but it may point to an empty string. `RecordPtr` is the same as the `widgRec` argument to `Tk_ConfigureWidget`; it points to the start of the widget record to modify. The last argument, `offset`, gives the offset in bytes from the start of the widget record to the location where the option value is to be placed. The procedure should translate the string to whatever form is appropriate for the option and store the value in the widget record. It should normally return TCL_OK, but if an error occurs in translating the string to a value then it should return TCL_ERROR and store an error message in `interp->result`.

The `printProc` procedure is called by `Tk_ConfigureInfo` to produce a string value describing an existing option. Its `clientData`, `tkwin`, `widgRec`, and `offset` arguments all have the same meaning as for `Tk_OptionParseProc` procedures. The `printProc` procedure should examine the option whose value is stored at `offset` in `widgRec`, produce a string describing that option, and return a pointer to the string. If the string is stored in dynamically-allocated memory, then the procedure must set `freeProcPtr` to the address of a procedure to call to free the string's memory; `Tk_ConfigureInfo` will call this procedure when it is finished with the string. If the result string is stored in static memory then `printProc` need not do anything with the `freeProcPtr` argument.

Once `parseProc` and `printProc` have been defined and a Tk_CustomOption structure has been created for them, options of this new type may be manipulated with Tk_ConfigSpec entries whose `type` fields are `TK_CONFIG_CUSTOM` and whose `customPtr` fields point to the Tk_CustomOption structure.

**EXAMPLES**

Although the explanation of `Tk_ConfigureWidget` is fairly complicated, its actual use is pretty straightforward. The easiest way to get started is to copy the code from an existing widget. The library implementation of frames (tkFrame.c) has a simple configuration table, and the library implementation of buttons (tkButton.c) has a much more complex table that uses many of the fancy `specFlags` mechanisms.

**SEE ALSO**

`Tk_SetOptions`

**KEYWORDS**

`anchor`, `bitmap`, `boolean`, `border`, `cap style`, `color`, `configuration options`, `cursor`, `custom`, `double`, `font`, `integer`, `join style`, `justify`, `millimeters`, `pixels`, `relief`, `synonym`, `uid`
ConfigWind

NAME

Tk_ConfigureWindow, Tk_MoveWindow, Tk_ResizeWindow, Tk_MoveResizeWindow,
Tk_SetWindowBorderWidth, Tk_ChangeWindowAttributes, Tk_SetWindowBackground,
Tk_SetWindowBackgroundPixmap, Tk_SetWindowBorder, Tk_SetWindowBorderPixmap,
Tk_SetWindowColormap, Tk_DefineCursor, Tk.UndefineCursor – change window configuration or
attributes

SYNOPSIS

#include <tk.h>
Tk_ConfigureWindow(tkwin, valueMask, valuePtr)
Tk_MoveWindow(tkwin, x, y)
Tk_ResizeWindow(tkwin, width, height)
Tk_MoveResizeWindow(tkwin, x, y, width, height)
Tk_SetWindowBorderWidth(tkwin, borderWidth)
Tk_ChangeWindowAttributes(tkwin, valueMask, attsPtr)
Tk_SetWindowBackground(tkwin, pixel)
Tk_SetWindowBackgroundPixmap(tkwin, pixmap)
Tk_SetWindowBorder(tkwin, pixel)
Tk_SetWindowBorderPixmap(tkwin, pixmap)
Tk_SetWindowColormap(tkwin, colormap)
Tk_DefineCursor(tkwin, cursor)
Tk.UndefineCursor(tkwin)

ARGUMENTS

Tk_Window tkwin (in)
Token for window.
unsigned int valueMask (in)
OR−ed mask of values like CWX or CWBorderPixel, indicating which fields of *valuePtr or
*attsPtr to use.
XWindowChanges *valuePtr (in)
Points to a structure containing new values for the configuration parameters selected by valueMask. Fields not selected by valueMask are ignored.
int x (in)
New x−coordinate for tkwin's top left pixel (including border, if any) within tkwin's parent.
int y (in)
New y−coordinate for tkwin's top left pixel (including border, if any) within tkwin's parent.
int width (in)
New width for tkwin (interior, not including border).
int height (in)
New height for tkwin (interior, not including border).

\textbf{int} \texttt{borderWidth (in)}

New width for tkwin's border.

\textbf{XSetWindowAttributes *attsPtr (in)}

Points to a structure containing new values for the attributes given by the \texttt{valueMask} argument. Attributes not selected by \texttt{valueMask} are ignored.

\textbf{unsigned long pixel (in)}

New background or border color for window.

\textbf{Pixmap pixmap (in)}

New pixmap to use for background or border of tkwin. WARNING: cannot necessarily be deleted immediately, as for Xlib calls. See note below.

\textbf{Colormap colormap (in)}

New colormap to use for tkwin.

\textbf{Tk_Cursor cursor (in)}

New cursor to use for tkwin. If \texttt{None} is specified, then tkwin will not have its own cursor; it will use the cursor of its parent.

\section*{DESCRIPTION}

These procedures are analogous to the X library procedures with similar names, such as \texttt{XConfigureWindow}. Each one of the above procedures calls the corresponding X procedure and also saves the configuration information in Tk's local structure for the window. This allows the information to be retrieved quickly by the application (using macros such as \texttt{Tk_X} and \texttt{Tk_Height}) without having to contact the X server. In addition, if no X window has actually been created for tkwin yet, these procedures do not issue X operations or cause event handlers to be invoked; they save the information in Tk's local structure for the window; when the window is created later, the saved information will be used to configure the window.

See the X library documentation for details on what these procedures do and how they use their arguments.

In the procedures \texttt{Tk_ConfigureWindow}, \texttt{Tk_MoveWindow}, \texttt{Tk_ResizeWindow}, \texttt{Tk_MoveResizeWindow}, and \texttt{Tk_SetWindowBorderWidth}, if tkwin is an internal window then event handlers interested in configure events are invoked immediately, before the procedure returns. If tkwin is a top-level window then the event handlers will be invoked later, after X has seen the request and returned an event for it.

Applications using Tk should never call procedures like \texttt{XConfigureWindow} directly; they should always use the corresponding Tk procedures.

The size and location of a window should only be modified by the appropriate geometry manager for that window and never by a window itself (but see \texttt{Tk_MoveToplevelWindow} for moving a top–level window).

You may not use \texttt{Tk_ConfigureWindow} to change the stacking order of a window (\texttt{valueMask} may not contain the \texttt{CWSibling} or \texttt{CWStackMode} bits). To change the stacking order, use the procedure \texttt{Tk_RestackWindow}.  

\section*{DESCRIPTION}
The procedure `Tk_SetWindowColormap` will automatically add `tkwin` to the `TK_COLORMAP_WINDOWS` property of its nearest top−level ancestor if the new colormap is different from that of `tkwin`'s parent and `tkwin` isn't already in the `TK_COLORMAP_WINDOWS` property.

**BUGS**

`Tk_SetWindowBackgroundPixmap` and `Tk_SetWindowBorderPixmap` differ slightly from their Xlib counterparts in that the `pixmap` argument may not necessarily be deleted immediately after calling one of these procedures. This is because `tkwin`'s window may not exist yet at the time of the call, in which case `pixmap` is merely saved and used later when `tkwin`'s window is actually created. If you wish to delete `pixmap`, then call `Tk_MakeWindowExist` first to be sure that `tkwin`'s window exists and `pixmap` has been passed to the X server.

A similar problem occurs for the `cursor` argument passed to `Tk_DefineCursor`. The solution is the same as for pixmaps above: call `Tk_MakeWindowExist` before freeing the cursor.

**SEE ALSO**

`Tk_MoveToplevelWindow`, `Tk_RestackWindow`

**KEYWORDS**

`attributes`, `border`, `color`, `configure`, `height`, `pixel`, `pixmap`, `width`, `window`, `x`, `y`
CoordToWin

NAME

Tk_CoordsToWindow – Find window containing a point

SYNOPSIS

#include <tk.h>
Tk_Window
Tk_CoordsToWindow(rootX, rootY, tkwin)

ARGUMENTS

int rootX (in)
   X−coordinate (in root window coordinates).
int rootY (in)
   Y−coordinate (in root window coordinates).
Tk_Window tkwin (in)
   Token for window that identifies application.

DESCRIPTION

Tk_CoordsToWindow locates the window that contains a given point. The point is specified in root
coordinates with rootX and rootY (if a virtual−root window manager is in use then rootX and rootY are in the
coordinate system of the virtual root window). The return value from the procedure is a token for the window
that contains the given point. If the point is not in any window, or if the containing window is not in the same
application as tkwin, then NULL is returned.

The containing window is decided using the same rules that determine which window contains the mouse
cursor: if a parent and a child both contain the point then the child gets preference, and if two siblings both
contain the point then the highest one in the stacking order (i.e. the one that's visible on the screen) gets
preference.

KEYWORDS

containing, coordinates, root window
BindTable

NAME

Tk_CreateBindingTable, Tk_DeleteBindingTable, Tk_CreateBinding, Tk_DeleteBinding, Tk_GetBinding, Tk_GetAllBindings, Tk_DeleteAllBindings, Tk_BindEvent – invoke scripts in response to X events

SYNOPSIS

#include <tk.h>

Tk_BindingTable Tk_CreateBindingTable(interp)

Tk_DeleteBindingTable(bindingTable)

unsigned long Tk_CreateBinding(interp, bindingTable, object, eventString, script, append)

int Tk_DeleteBinding(interp, bindingTable, object, eventString)

CONST char * Tk_GetBinding(interp, bindingTable, object, eventString)

Tk_GetAllBindings(interp, bindingTable, object)

Tk_DeleteAllBindings(bindingTable, object)

Tk_BindEvent(bindingTable, eventPtr, tkwin, numObjects, objectPtr)

ARGUMENTS

Tcl_Interp *interp (in)

Interpreter to use when invoking bindings in binding table. Also used for returning results and errors from binding procedures.

Tk_BindingTable bindingTable (in)

Token for binding table; must have been returned by some previous call to Tk_CreateBindingTable.

ClientData object (in)

Identifies object with which binding is associated.

CONST char *eventString (in)

String describing event sequence.

char *script (in)

Tcl script to invoke when binding triggers.

int append (in)

Non-zero means append script to existing script for binding, if any; zero means replace existing script with new one.

XEvent *eventPtr (in)

X event to match against bindings in bindingTable.

Tk_Window tkwin (in)

Identifier for any window on the display where the event occurred. Used to find display-related information such as key maps.
DESCRIPTION

These procedures provide a general-purpose mechanism for creating and invoking bindings. Bindings are organized in terms of binding tables. A binding table consists of a collection of bindings plus a history of recent events. Within a binding table, bindings are associated with objects. The meaning of an object is defined by clients of the binding package. For example, Tk keeps uses one binding table to hold all of the bindings created by the bind command. For this table, objects are pointers to strings such as window names, class names, or other binding tags such as all. Tk also keeps a separate binding table for each canvas widget, which manages bindings created by the canvas's bind widget command; within this table, an object is either a pointer to the internal structure for a canvas item or a Tk_Uid identifying a tag.

The procedure Tk_CreateBindingTable creates a new binding table and associates interp with it (when bindings in the table are invoked, the scripts will be evaluated in interp). Tk_CreateBindingTable returns a token for the table, which must be used in calls to other procedures such as Tk_CreateBinding or Tk_BindEvent.

Tk_DeleteBindingTable frees all of the state associated with a binding table. Once it returns the caller should not use the bindingTable token again.

Tk_CreateBinding adds a new binding to an existing table. The object argument identifies the object with which the binding is to be associated, and it may be any one-word value. Typically it is a pointer to a string or data structure. The eventString argument identifies the event or sequence of events for the binding; see the documentation for the bind command for a description of its format. script is the Tcl script to be evaluated when the binding triggers. append indicates what to do if there already exists a binding for object and eventString: if append is zero then script replaces the old script; if append is non-zero then the new script is appended to the old one. Tk_CreateBinding returns an X event mask for all the events associated with the bindings. This information may be useful to invoke XSelectInput to select relevant events, or to disallow the use of certain events in bindings. If an error occurred while creating the binding (e.g., eventString refers to a non-existent event), then 0 is returned and an error message is left in interp->result.

Tk_DeleteBinding removes from bindingTable the binding given by object and eventString, if such a binding exists. Tk_DeleteBinding always returns TCL_OK. In some cases it may reset interp->result to the default empty value.

Tk_GetBinding returns a pointer to the script associated with eventString and object in bindingTable. If no such binding exists then NULL is returned and an error message is left in interp->result.

Tk_GetAllBindings returns in interp->result a list of all the event strings for which there are bindings in bindingTable associated with object. If there are no bindings for object then an empty string is returned in
interp->result.

Tk_DeleteAllBindings deletes all of the bindings in bindingTable that are associated with object.

Tk_BindEvent is called to process an event. It makes a copy of the event in an internal history list associated with the binding table, then it checks for bindings that match the event. Tk_BindEvent processes each of the objects pointed to by objectPtr in turn. For each object, it finds all the bindings that match the current event history, selects the most specific binding using the priority mechanism described in the documentation for bind, and invokes the script for that binding. If there are no matching bindings for a particular object, then the object is skipped. Tk_BindEvent continues through all of the objects, handling exceptions such as errors, break, and continue as described in the documentation for bind.

KEYWORDS

binding, event, object, script

Copyright ' 1994 The Regents of the University of California.  
Copyright ' 1994-1996 Sun Microsystems, Inc.  
Copyright ' 1995-1997 Roger E. Critchlow Jr.
CrtCmHdlr

NAME

Tk_CreateClientMessageHandler, Tk_DeleteClientMessageHandler – associate procedure callback with
ClientMessage type X events

SYNOPSIS

#include <tk.h>
Tk_CreateClientMessageHandler(proc)
Tk_DeleteClientMessageHandler(proc)

ARGUMENTS

Tk_ClientMessageProc *proc (in)
   Procedure to invoke whenever a ClientMessage X event occurs on any display.

DESCRIPTION

Tk_CreateClientMessageHandler arranges for proc to be invoked in the future whenever a ClientMessage
X event occurs that isn't handled by WM_PROTOCOL. Tk_CreateClientMessageHandler is intended for
use by applications which need to watch X ClientMessage events, such as drag and drop applications.

The callback to proc will be made by Tk_HandleEvent; this mechanism only works in programs that
dispatch events through Tk_HandleEvent (or through other Tk procedures that call Tk_HandleEvent, such
as Tk_DoOneEvent or Tk_MainLoop).

Proc should have arguments and result that match the type Tk_ClientMessageProc:

typedef int Tk_ClientMessageProc(
    Tk_Window tkwin,
    XEvent *eventPtr);

The tkwin parameter to proc is the Tk window which is associated with this event. EventPtr is a pointer to the
X event.

Whenever an X ClientMessage event is processed by Tk_HandleEvent, the proc is called if it wasn't handled
as a WM_PROTOCOL. The return value from proc is normally 0. A non–zero return value indicates that the
event is not to be handled further; that is, proc has done all processing that is to be allowed for the event.

If there are multiple ClientMessage event handlers, each one is called for each event, in the order in which
they were established.
Tk_DeleteClientMessageHandler may be called to delete a previously-created ClientMessage event handler: it deletes each handler it finds that matches the proc argument. If no such handler exists, then Tk_DeleteClientMessageHandler returns without doing anything. Although Tk supports it, it's probably a bad idea to have more than one callback with the same proc argument.

**KEYWORDS**

bind, callback, event, handler

---

Copyright ' 2000 Ajuba Solutions.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
CrtErrHdlr

NAME

Tk_CreateErrorHandler, Tk_DeleteErrorHandler – handle X protocol errors

SYNOPSIS

#include <tk.h>
Tk_ErrorHandler
Tk_CreateErrorHandler(display, error, request, minor, proc, clientData)
Tk_DeleteErrorHandler(handler)

ARGUMENTS

Display *display (in)
    Display whose errors are to be handled.
int error (in)
    Match only error events with this value in the error_code field. If −1, then match any error_code value.
int request (in)
    Match only error events with this value in the request_code field. If −1, then match any request_code value.
int minor (in)
    Match only error events with this value in the minor_code field. If −1, then match any minor_code value.
Tk_ErrorProc *proc (in)
    Procedure to invoke whenever an error event is received for display and matches error, request, and minor. NULL means ignore any matching errors.
ClientData clientData (in)
    Arbitrary one-word value to pass to proc.
Tk_ErrorHandler handler (in)
    Token for error handler to delete (return value from a previous call to Tk_CreateErrorHandler).

DESCRIPTION

Tk_CreateErrorHandler arranges for a particular procedure (proc) to be called whenever certain protocol errors occur on a particular display (display). Protocol errors occur when the X protocol is used incorrectly, such as attempting to map a window that doesn't exist. See the Xlib documentation for XSetErrorHandler for more information on the kinds of errors that can occur. For proc to be invoked to handle a particular error, five things must occur:

[1] The error must pertain to display.
Either the *error* argument to **Tk_CreateErrorHandler** must have been −1, or the *error* argument must match the *error_code* field from the error event.

Either the *request* argument to **Tk_CreateErrorHandler** must have been −1, or the *request* argument must match the *request_code* field from the error event.

Either the *minor* argument to **Tk_CreateErrorHandler** must have been −1, or the *minor* argument must match the *minor_code* field from the error event.

The protocol request to which the error pertains must have been made when the handler was active (see below for more information).

**Proc** should have arguments and result that match the following type:

```c
typedef int Tk_ErrorProc(
    ClientData clientData,
    XErrorEvent *errEventPtr);
```

The *clientData* parameter to *proc* is a copy of the *clientData* argument given to **Tcl_CreateErrorHandler** when the callback was created. Typically, *clientData* points to a data structure containing application−specific information that is needed to deal with the error. *ErrEventPtr* is a pointer to the X error event. The procedure *proc* should return an integer value. If it returns 0 it means that *proc* handled the error completely and there is no need to take any other action for the error. If it returns non-zero it means *proc* was unable to handle the error.

If a value of NULL is specified for *proc*, all matching errors will be ignored: this will produce the same result as if a procedure had been specified that always returns 0.

If more than more than one handler matches a particular error, then they are invoked in turn. The handlers will be invoked in reverse order of creation: most recently declared handler first. If any handler returns 0, then subsequent (older) handlers will not be invoked. If no handler returns 0, then Tk invokes X'es default error handler, which prints an error message and aborts the program. If you wish to have a default handler that deals with errors that no other handler can deal with, then declare it first.

The X documentation states that "the error handler should not call any functions (directly or indirectly) on the display that will generate protocol requests or that will look for input events." This restriction applies to handlers declared by **Tk_CreateErrorHandler**; disobey it at your own risk.

**Tk_DeleteErrorHandler** may be called to delete a previously−created error handler. The *handler* argument identifies the error handler, and should be a value returned by a previous call to **Tk_CreateEventHandler**.

A particular error handler applies to errors resulting from protocol requests generated between the call to **Tk_CreateErrorHandler** and the call to **Tk_DeleteErrorHandler**. However, the actual callback to *proc* may not occur until after the **Tk_DeleteErrorHandler** call, due to buffering in the client and server. If an error event pertains to a protocol request made just before calling **Tk_DeleteErrorHandler**, then the error
event may not have been processed before the `Tk_DeleteErrorHandler` call. When this situation arises, Tk will save information about the handler and invoke the handler's proc later when the error event finally arrives. If an application wishes to delete an error handler and know for certain that all relevant errors have been processed, it should first call `Tk_DeleteErrorHandler` and then call `XSync`; this will flush out any buffered requests and errors, but will result in a performance penalty because it requires communication to and from the X server. After the `XSync` call Tk is guaranteed not to call any error handlers deleted before the `XSync` call.

For the Tk error handling mechanism to work properly, it is essential that application code never calls `XSetErrorHandler` directly; applications should use only `Tk_CreateErrorHandler`.

**KEYWORDS**

callback, error, event, handler

---

Copyright ' 1990 The Regents of the University of California.  
Copyright ' 1994−1996 Sun Microsystems, Inc.  
Copyright ' 1995−1997 Roger E. Critchlow Jr.
EventHndlr

NAME

Tk_CreateEventHandler, Tk_DeleteEventHandler – associate procedure callback with an X event

SYNOPSIS

#include <tk.h>
Tk_CreateEventHandler(tkwin, mask, proc, clientData)
Tk_DeleteEventHandler(tkwin, mask, proc, clientData)

ARGUMENTS

Tk_Window tkwin (in)
   Token for window in which events may occur.
unsigned long mask (in)
   Bit−mask of events (such as ButtonPressMask) for which proc should be called.
Tk_EventProc *proc (in)
   Procedure to invoke whenever an event in mask occurs in the window given by tkwin.
ClientData clientData (in)
   Arbitrary one−word value to pass to proc.

DESCRIPTION

Tk_CreateEventHandler arranges for proc to be invoked in the future whenever one of the event types specified by mask occurs in the window specified by tkwin. The callback to proc will be made by Tk_HandleEvent; this mechanism only works in programs that dispatch events through Tk_HandleEvent (or through other Tk procedures that call Tk_HandleEvent, such as Tk_DoOneEvent or Tk_MainLoop).

Proc should have arguments and result that match the type Tk_EventProc:

typedef void Tk_EventProc(
    ClientData,
    XEvent *);

The clientData parameter to proc is a copy of the clientData argument given to Tk_CreateEventHandler when the callback was created. Typically, clientData points to a data structure containing application−specific information about the window in which the event occurred. EventPtr is a pointer to the X event, which will be one of the ones specified in the mask argument to Tk_CreateEventHandler.

Tk_DeleteEventHandler may be called to delete a previously−created event handler: it deletes the first handler it finds that is associated with tkwin and matches the mask, proc, and clientData arguments. If no such handler exists, then Tk_HandleEvent returns without doing anything. Although Tk supports it, it's probably a bad idea to have more than one callback with the same mask, proc, and clientData arguments. When a
window is deleted all of its handlers will be deleted automatically; in this case there is no need to call Tk_DeleteEventHandler.

If multiple handlers are declared for the same type of X event on the same window, then the handlers will be invoked in the order they were created.

KEYWORDS

bind, callback, event, handler

Copyright © 1990 The Regents of the University of California.
Copyright © 1994−1996 Sun Microsystems, Inc.
Copyright © 1995−1997 Roger E. Critchlow Jr.
CrtGenHdlr

NAME

Tk_CreateGenericHandler, Tk_DeleteGenericHandler – associate procedure callback with all X events

SYNOPSIS

```
#include <tk.h>
Tk_CreateGenericHandler(proc, clientData)
Tk_DeleteGenericHandler(proc, clientData)
```

ARGUMENTS

- `Tk_GenericProc *proc (in)`
  - Procedure to invoke whenever any X event occurs on any display.
- `ClientData clientData (in)`
  - Arbitrary one-word value to pass to `proc`.

DESCRIPTION

`Tk_CreateGenericHandler` arranges for `proc` to be invoked in the future whenever any X event occurs. This mechanism is *not* intended for dispatching X events on windows managed by Tk (you should use `Tk_CreateEventHandler` for this purpose). `Tk_CreateGenericHandler` is intended for other purposes, such as tracing X events, monitoring events on windows not owned by Tk, accessing X-related libraries that were not originally designed for use with Tk, and so on.

The callback to `proc` will be made by `Tk_HandleEvent`; this mechanism only works in programs that dispatch events through `Tk_HandleEvent` (or through other Tk procedures that call `Tk_HandleEvent`, such as `Tk_DoOneEvent` or `Tk_MainLoop`).

`Proc` should have arguments and result that match the type `Tk_GenericProc`:

```
typedef int Tk_GenericProc(
    ClientData clientData,
    XEvent *eventPtr);
```

The `clientData` parameter to `proc` is a copy of the `clientData` argument given to `Tk_CreateGenericHandler` when the callback was created. Typically, `clientData` points to a data structure containing application-specific information about how to handle events. `EventPtr` is a pointer to the X event.

Whenever an X event is processed by `Tk_HandleEvent`, `proc` is called. The return value from `proc` is normally 0. A non-zero return value indicates that the event is not to be handled further; that is, `proc` has done all processing that is to be allowed for the event.
If there are multiple generic event handlers, each one is called for each event, in the order in which they were established.

**Tk_DeleteGenericHandler** may be called to delete a previously-created generic event handler: it deletes each handler it finds that matches the `proc` and `clientData` arguments. If no such handler exists, then **Tk_DeleteGenericHandler** returns without doing anything. Although Tk supports it, it's probably a bad idea to have more than one callback with the same `proc` and `clientData` arguments.

Establishing a generic event handler does nothing to ensure that the process will actually receive the X events that the handler wants to process. For example, it is the caller's responsibility to invoke **XSelectInput** to select the desired events, if that is necessary.

**KEYWORDS**

`bind`, `callback`, `event`, `handler`
CrtImgType

NAME

Tk_CreateImageType, Tk_GetImageMasterData, Tk_InitImageArgs – define new kind of image

SYNOPSIS

ARGUMENTS

NAME

PORTABILITY

CREATEPROC

GETPROC

DISPLAYPROC

FREEPROC

DELETEPROC

TK_GETIMAGEMASTERDATA

TK_INITIMAGEARGS

SEE ALSO

KEYWORDS

NAME

Tk_CreateImageType, Tk_GetImageMasterData, Tk_InitImageArgs – define new kind of image

SYNOPSIS

#include <tk.h>

Tk_CreateImageType(typePtr)

ClientData

Tk_GetImageMasterData(interp, name, typePtrPtr)

Tk_InitImageArgs(interp, argc, argvPtr)

ARGUMENTS

Tk_ImageType *typePtr (in)

Structure that defines the new type of image. Must be static: a pointer to this structure is retained by
the image code.

Tcl_Interp *interp (in)

Interpreter in which image was created.

CONST char *name (in)

Name of existing image.

Tk_ImageType **typePtrPtr (out)
Points to word in which to store a pointer to type information for the given image, if it exists.

```
int argc (in)
Number of arguments
```

```
char ***argvPtr (in/out)
Pointer to argument list
```

**DESCRIPTION**

**Tk_CreateImageType** is invoked to define a new kind of image. An image type corresponds to a particular value of the *type* argument for the **image create** command. There may exist any number of different image types, and new types may be defined dynamically by calling **Tk_CreateImageType**. For example, there might be one type for 2−color bitmaps, another for multi−color images, another for dithered images, another for video, and so on.

The code that implements a new image type is called an *image manager*. It consists of a collection of procedures plus three different kinds of data structures. The first data structure is a Tk_ImageType structure, which contains the name of the image type and pointers to five procedures provided by the image manager to deal with images of this type:

```
typedef struct Tk_ImageType {
    char *name;
    Tk_ImageCreateProc *createProc;
    Tk_ImageGetProc *getProc;
    Tk_ImageDisplayProc *displayProc;
    Tk_ImageFreeProc *freeProc;
    Tk_ImageDeleteProc *deleteProc;
} Tk_ImageType;
```

The fields of this structure will be described in later subsections of this entry.

The second major data structure manipulated by an image manager is called an *image master*; it contains overall information about a particular image, such as the values of the configuration options specified in an **image create** command. There will usually be one of these structures for each invocation of the **image create** command.

The third data structure related to images is an *image instance*. There will usually be one of these structures for each usage of an image in a particular widget. It is possible for a single image to appear simultaneously in multiple widgets, or even multiple times in the same widget. Furthermore, different instances may be on different screens or displays. The image instance data structure describes things that may vary from instance to instance, such as colors and graphics contexts for redisplay. There is usually one instance structure for each **−image** option specified for a widget or canvas item.

The following subsections describe the fields of a Tk_ImageType in more detail.
NAME

typePtr−>name provides a name for the image type. Once Tk_CreateImageType returns, this name may be used in image create commands to create images of the new type. If there already existed an image type by this name then the new image type replaces the old one.

PORTABILITY

In Tk 8.2 and earlier, the createProc below had a different signature. If you want to compile an image type using the old interface which should still run on all Tcl/Tk versions, compile it with the flag −DUSE_OLD_IMAGE. Further on, if you are using Stubs, you need to call the function Tk_InitImageArgs(interp, argc, &argv) first in your createProc. See below for a description of this function.

CREATEPROC

typePtr−>createProc provides the address of a procedure for Tk to call whenever image create is invoked to create an image of the new type. typePtr−>createProc must match the following prototype:

typedef int Tk_ImageCreateProc(
    Tcl_Interp *interp,
    const char *name,
    int objc,
    Tcl_Obj *CONST objv[],
    Tk_ImageType *typePtr,
    Tk_ImageMaster master,
    ClientData *masterDataPtr);

The interp argument is the interpreter in which the image command was invoked, and name is the name for the new image, which was either specified explicitly in the image command or generated automatically by the image command. The objc and objv arguments describe all the configuration options for the new image (everything after the name argument to image). The master argument is a token that refers to Tk's information about this image; the image manager must return this token to Tk when invoking the Tk_ImageChanged procedure. Typically createProc will parse objc and objv and create an image master data structure for the new image. createProc may store an arbitrary one-word value at masterDataPtr, which will be passed back to the image manager when other callbacks are invoked. Typically the value is a pointer to the master data structure for the image.

If createProc encounters an error, it should leave an error message in interp−>result and return TCL_ERROR; otherwise it should return TCL_OK.

createProc should call Tk_ImageChanged in order to set the size of the image and request an initial redisplay.
GETPROC

typePtr->getProc is invoked by Tk whenever a widget calls \texttt{Tk\_GetImage} to use a particular image. This procedure must match the following prototype:

\begin{verbatim}
typedef ClientData Tk\_ImageGetProc(
    Tk\_Window tkwin,
    ClientData masterData);
\end{verbatim}

The \texttt{tkwin} argument identifies the window in which the image will be used and \texttt{masterData} is the value returned by \texttt{createProc} when the image master was created. \texttt{getProc} will usually create a data structure for the new instance, including such things as the resources needed to display the image in the given window. \texttt{getProc} returns a one-word token for the instance, which is typically the address of the instance data structure. Tk will pass this value back to the image manager when invoking its \texttt{displayProc} and \texttt{freeProc} procedures.

DISPLAYPROC

typePtr->displayProc is invoked by Tk whenever an image needs to be displayed (i.e., whenever a widget calls \texttt{Tk\_RedrawImage}). \texttt{displayProc} must match the following prototype:

\begin{verbatim}
typedef void Tk\_ImageDisplayProc(
    ClientData instanceData,
    Display *display,
    Drawable drawable,
    int imageX,
    int imageY,
    int width,
    int height,
    int drawableX,
    int drawableY);
\end{verbatim}

The \texttt{instanceData} will be the same as the value returned by \texttt{getProc} when the instance was created. \texttt{display} and \texttt{drawable} indicate where to display the image; \texttt{drawable} may be a pixmap rather than the window specified to \texttt{getProc} (this is usually the case, since most widgets double-buffer their redisplay to get smoother visual effects). \texttt{imageX}, \texttt{imageY}, \texttt{width}, and \texttt{height} identify the region of the image that must be redisplayed. This region will always be within the size of the image as specified in the most recent call to \texttt{Tk\_ImageChanged}. \texttt{drawableX} and \texttt{drawableY} indicate where in \texttt{drawable} the image should be displayed; \texttt{displayProc} should display the given region of the image so that point \texttt{(imageX, imageY)} in the image appears at \texttt{(drawableX, drawableY)} in \texttt{drawable}.

FREEPROC

typePtr->freeProc contains the address of a procedure that Tk will invoke when an image instance is released (i.e., when \texttt{Tk\_FreImage} is invoked). This can happen, for example, when a widget is deleted or a image item in a canvas is deleted, or when the image displayed in a widget or canvas item is changed. \texttt{freeProc} must match the following prototype:
typedef void Tk_ImageFreeProc(
    ClientData instanceData,
    Display *display);

The instanceData will be the same as the value returned by proc when the instance was created, and display is the display containing the window for the instance. freeProc should release any resources associated with the image instance, since the instance will never be used again.

DELETEPROC

typePtr−>deleteProc is a procedure that Tk invokes when an image is being deleted (i.e. when the image delete command is invoked). Before invoking deleteProc Tk will invoke freeProc for each of the image’s instances. deleteProc must match the following prototype:

typedef void Tk_ImageDeleteProc(
    ClientData masterData);

The masterData argument will be the same as the value stored in *masterDataPtr by createProc when the image was created. deleteProc should release any resources associated with the image.

TK_GETIMAGEMASTERDATA

The procedure Tk_GetImageMasterData may be invoked to retrieve information about an image. For example, an image manager can use this procedure to locate its image master data for an image. If there exists an image named name in the interpreter given by interp, then *typePtrPtr is filled in with type information for the image (the typePtr value passed to Tk_CreateImageType when the image type was registered) and the return value is the ClientData value returned by the createProc when the image was created (this is typically a pointer to the image master data structure). If no such image exists then NULL is returned and NULL is stored at *typePtrPtr.

TK_INITIMAGEARGS

The function Tk_InitImageArgs converts the arguments of the createProc from objects to strings when necessary. When not using stubs, not using the old interface, or running under an older (pre−8.3) Tk version, this function has no effect. This function makes porting older image handlers to the new interface a lot easier: After running this function, the arguments are guaranteed to be in string format, no matter how Tk deliverd them.

SEE ALSO

Tk_ImageChanged, Tk_GetImage, Tk_FreeImage, Tk_RedrawImage, Tk_SizeOfImage
KEYWORDS

image manager, image type, instance, master

Copyright ' 1994 The Regents of the University of California.
Copyright ' 1994-1997 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

Tk_CreateItemType, Tk_GetItemTypes – define new kind of canvas item

SYNOPSIS

#include <tk.h>

Tk_CreateItemType(typePtr)

Tk_ItemType *

Tk_GetItemTypes()

ARGUMENTS

INTRODUCTION

DATA STRUCTURES

NAME

ITEMSIZE

CREATEPROC

CONFIGSPECS

CONFIGPROC

COORDPROC

DELETEPROC

DISPLAYPROC AND ALWAYSREDRAW

POINTPROC

AREAPROC

POSTSCRIPTPROC

SCALEPROC

TRANSLATEPROC

INDEXPROC

ICURSORPROC

SELECTIONPROC

INSERTPROC

DCHARSPROC

SEE ALSO

KEYWORDS
ARGUMENTS

Tk_ItemType *typePtr (in)
    Structure that defines the new type of canvas item.

INTRODUCTION

Tk_CreateItemType is invoked to define a new kind of canvas item described by the typePtr argument. An item type corresponds to a particular value of the type argument to the create widget command for canvases, and the code that implements a canvas item type is called a type manager. Tk defines several built-in item types, such as rectangle and text and image, but Tk_CreateItemType allows additional item types to be defined. Once Tk_CreateItemType returns, the new item type may be used in new or existing canvas widgets just like the built-in item types.

Tk_GetItemTypes returns a pointer to the first in the list of all item types currently defined for canvases. The entries in the list are linked together through their nextPtr fields, with the end of the list marked by a NULL nextPtr.

Tk provides a number of utility procedures for the use of canvas type managers, such as Tk_CanvasCoords and Tk_CanvasPsColor; these are described in separate manual entries.

DATA STRUCTURES

A type manager consists of a collection of procedures that provide a standard set of operations on items of that type. The type manager deals with three kinds of data structures. The first data structure is a Tk_ItemType; it contains information such as the name of the type and pointers to the standard procedures implemented by the type manager:

```
typedef struct Tk_ItemType {
    char *name;
    int itemSize;
    Tk_ItemCreateProc *createProc;
    Tk_ConfigSpec *configSpecs;
    Tk_ItemConfigureProc *configProc;
    Tk_ItemCoordProc *coordProc;
    Tk_ItemDeleteProc *deleteProc;
    Tk_ItemDisplayProc *displayProc;
    int alwaysRedraw;
    Tk_ItemPointProc *pointProc;
    Tk_ItemAreaProc *areaProc;
    Tk_ItemPostscriptProc *postscriptProc;
    Tk_ItemScaleProc *scaleProc;
    Tk_ItemTranslateProc *translateProc;
    Tk_ItemIndexProc *indexProc;
} Tk_ItemType;
```
Tk_ItemCursorProc *
icursorProc;
Tk_ItemSelectionProc *
selectionProc;
Tk_ItemInsertProc *
insertProc;
Tk_ItemDCharsProc *
dCharsProc;
Tk_ItemType *
nextPtr;
} Tk_ItemType;

The fields of a Tk_ItemType structure are described in more detail later in this manual entry. When Tk_CreateItemType is called, its typePtr argument must point to a structure with all of the fields initialized except nextPtr, which Tk sets to link all the types together into a list. The structure must be in permanent memory (either statically allocated or dynamically allocated but never freed); Tk retains a pointer to this structure.

The second data structure manipulated by a type manager is an item record. For each item in a canvas there exists one item record. All of the items of a given type generally have item records with the same structure, but different types usually have different formats for their item records. The first part of each item record is a header with a standard structure defined by Tk via the type Tk_Item; the rest of the item record is defined by the type manager. A type manager must define its item records with a Tk_Item as the first field. For example, the item record for bitmap items is defined as follows:

```c
typedef struct BitmapItem {
    Tk_Item header;
    double x, y;
    Tk_Anchor anchor;
    Pixmap bitmap;
    XColor *fgColor;
    XColor *bgColor;
    GC gc;
} BitmapItem;
```

The header substructure contains information used by Tk to manage the item, such as its identifier, its tags, its type, and its bounding box. The fields starting with x belong to the type manager: Tk will never read or write them. The type manager should not need to read or write any of the fields in the header except for four fields whose names are x1, y1, x2, and y2. These fields give a bounding box for the items using integer canvas coordinates: the item should not cover any pixels with x−coordinate lower than x1 or y−coordinate lower than y1, nor should it cover any pixels with x−coordinate greater than or equal to x2 or y−coordinate greater than or equal to y2. It is up to the type manager to keep the bounding box up to date as the item is moved and reconfigured.

Whenever Tk calls a procedure in a type manager it passes in a pointer to an item record. The argument is always passed as a pointer to a Tk_Item; the type manager will typically cast this into a pointer to its own specific type, such as BitmapItem.

The third data structure used by type managers has type Tk_Canvas; it serves as an opaque handle for the canvas widget as a whole. Type managers need not know anything about the contents of this structure. A Tk_Canvas handle is typically passed in to the procedures of a type manager, and the type manager can pass the handle back to library procedures such as Tk_CanvasTkwin to fetch information about the canvas.
NAME

This section and the ones that follow describe each of the fields in a Tk_ItemType structure in detail. The name field provides a string name for the item type. Once Tk_CreateImageType returns, this name may be used in create widget commands to create items of the new type. If there already existed an item type by this name then the new item type replaces the old one.

ITEMSIZE

typePtr->itemSize gives the size in bytes of item records of this type, including the Tk_Item header. Tk uses this size to allocate memory space for items of the type. All of the item records for a given type must have the same size. If variable length fields are needed for an item (such as a list of points for a polygon), the type manager can allocate a separate object of variable length and keep a pointer to it in the item record.

CREATEPROC

typePtr->createProc points to a procedure for Tk to call whenever a new item of this type is created. typePtr->createProc must match the following prototype:

```
typedef int Tk_ItemCreateProc(
    Tcl_Interp *interp,
    Tk_Canvas *canvas,
    Tk_Item *itemPtr,
    int objc,
    Tcl_Obj* CONST objv);
```

The interp argument is the interpreter in which the canvas's create widget command was invoked, and canvas is a handle for the canvas widget. itemPtr is a pointer to a newly–allocated item of size typePtr->itemSize. Tk has already initialized the item's header (the first sizeof(Tk_ItemType) bytes). The objc and objv arguments describe all of the arguments to the create command after the type argument. For example, in the widget command

```
.c create rectangle 10 20 50 50 -fill black
```

objc will be 6 and objv[0] will contain the integer object 10.

createProc should use objc and objv to initialize the type–specific parts of the item record and set an initial value for the bounding box in the item's header. It should return a standard Tcl completion code and leave an error message in interp->result if an error occurs. If an error occurs Tk will free the item record, so createProc must be sure to leave the item record in a clean state if it returns an error (e.g., it must free any additional memory that it allocated for the item).

CONFIGSPECS

Each type manager must provide a standard table describing its configuration options, in a form suitable for use with Tk_ConfigureWidget. This table will normally be used by typePtr->createProc and
typePtr−>configProc, but Tk also uses it directly to retrieve option information in the itemcget and itemconfigure widget commands. typePtr−>configSpecs must point to the configuration table for this type. Note: Tk provides a custom option type tk_CanvasTagsOption for implementing the −tags option; see an existing type manager for an example of how to use it in configSpecs.

CONFIGPROC

typePtr−>configProc is called by Tk whenever the itemconfigure widget command is invoked to change the configuration options for a canvas item. This procedure must match the following prototype:

```c
typedef int Tk_ItemConfigureProc(
    Tcl_Interp *interp,
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    int objc,
    Tcl_Obj* CONST objv,
    int flags);
```

The interp objument identifies the interpreter in which the widget command was invoked, canvas is a handle for the canvas widget, and itemPtr is a pointer to the item being configured. objc and objv contain the configuration options. For example, if the following command is invoked:

```
.c itemconfigure 2 −fill red −outline black
```

objc is 4 and objv contains the string objects −fill through black. objc will always be an even value. The flags argument contains flags to pass to Tk_ConfigureWidget; currently this value is always TK_CONFIG_ARGV_ONLY when Tk invokes typePtr−>configProc, but the type manager's createProc procedure will usually invoke configProc with different flag values.

typePtr−>configProc returns a standard Tcl completion code and leaves an error message in interp−>result if an error occurs. It must update the item's bounding box to reflect the new configuration options.

COORDPROC

typePtr−>coordProc is invoked by Tk to implement the coords widget command for an item. It must match the following prototype:

```c
typedef int Tk_ItemCoordProc(
    Tcl_Interp *interp,
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    int objc,
    Tcl_Obj* CONST objv);
```

The arguments interp, canvas, and itemPtr all have the standard meanings, and objc and objv describe the coordinate arguments. For example, if the following widget command is invoked:

```
.c coords 2 30 90
```
objc will be 2 and objv will contain the integer objects 30 and 90.

The coordProc procedure should process the new coordinates, update the item appropriately (e.g., it must reset the bounding box in the item's header), and return a standard Tcl completion code. If an error occurs, coordProc must leave an error message in interp−>result.

DELETEPROC

typePtr−>deleteProc is invoked by Tk to delete an item and free any resources allocated to it. It must match the following prototype:

```c
typedef void Tk_ItemDeleteProc(
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    Display *display);
```

The canvas and itemPtr arguments have the usual interpretations, and display identifies the X display containing the canvas. deleteProc must free up any resources allocated for the item, so that Tk can free the item record. deleteProc should not actually free the item record; this will be done by Tk when deleteProc returns.

DISPLAYPROC AND ALWAYSREDRAW

```c
typedef void Tk_ItemDisplayProc(
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    Display *display,
    Drawable dst,
    int x,
    int y,
    int width,
    int height);
```

The canvas and itemPtr arguments have the usual meaning, display identifies the display containing the canvas, and dst specifies a drawable in which the item should be rendered; typically this is an off−screen pixmap, which Tk will copy into the canvas's window once all relevant items have been drawn. x, y, width, and height specify a rectangular region in canvas coordinates, which is the area to be redrawn; only information that overlaps this area needs to be redrawn. Tk will not call displayProc unless the item's bounding box overlaps the redraw area, but the type manager may wish to use the redraw area to optimize the redisplay of the item.

Because of scrolling and the use of off−screen pixmaps for double−buffered redisplay, the item's coordinates in dst will not necessarily be the same as those in the canvas. displayProc should call Tk_CanvasDrawableCoords to transform coordinates from those of the canvas to those of dst.
Normally an item's displayProc is only invoked if the item overlaps the area being displayed. However, if typePtr->alwaysRedraw has a non-zero value, then displayProc is invoked during every redisplay operation, even if the item doesn't overlap the area of redisplay. alwaysRedraw should normally be set to 0; it is only set to 1 in special cases such as window items that need to be unmapped when they are off-screen.

**POINTPROC**

typePtr->pointProc is invoked by Tk to find out how close a given point is to a canvas item. Tk uses this procedure for purposes such as locating the item under the mouse or finding the closest item to a given point. The procedure must match the following prototype:

```c
typedef double Tk_ItemPointProc(
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    double *pointPtr);
```

`canvas` and `itemPtr` have the usual meaning. `pointPtr` points to an array of two numbers giving the x and y coordinates of a point. `pointProc` must return a real value giving the distance from the point to the item, or 0 if the point lies inside the item.

**AREAPROC**

typePtr->areaProc is invoked by Tk to find out the relationship between an item and a rectangular area. It must match the following prototype:

```c
typedef int Tk_ItemAreaProc(
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    double *rectPtr);
```

`canvas` and `itemPtr` have the usual meaning. `rectPtr` points to an array of four real numbers; the first two give the x and y coordinates of the upper left corner of a rectangle, and the second two give the x and y coordinates of the lower right corner. `areaProc` must return −1 if the item lies entirely outside the given area, 0 if it lies partially inside and partially outside the area, and 1 if it lies entirely inside the area.

**POSTSCRIPTPROC**

typePtr->postscriptProc is invoked by Tk to generate Postscript for an item during the postscript widget command. If the type manager is not capable of generating Postscript then typePtr->postscriptProc should be NULL. The procedure must match the following prototype:

```c
typedef int Tk_ItemPostscriptProc(
    Tcl_Interp *interp,
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    int prepass);
```
The interp, canvas, and itemPtr arguments all have standard meanings; prepass will be described below. If postscriptProc completes successfully, it should append Postscript for the item to the information in interp->result (e.g. by calling Tcl_AppendResult, not Tcl_SetResult) and return TCL_OK. If an error occurs, postscriptProc should clear the result and replace its contents with an error message; then it should return TCL_ERROR.

Tk provides a collection of utility procedures to simplify postscriptProc. For example, Tk_CanvasPsColor will generate Postscript to set the current color to a given Tk color and Tk_CanvasPsFont will set up font information. When generating Postscript, the type manager is free to change the graphics state of the Postscript interpreter, since Tk places gsav and greset commands around the Postscript for the item. The type manager can use canvas x coordinates directly in its Postscript, but it must call Tk_CanvasPsY to convert y coordinates from the space of the canvas (where the origin is at the upper left) to the space of Postscript (where the origin is at the lower left).

In order to generate Postscript that complies with the Adobe Document Structuring Conventions, Tk actually generates Postscript in two passes. It calls each item's postscriptProc in each pass. The only purpose of the first pass is to collect font information (which is done by Tk_CanvasPsFont); the actual Postscript is discarded. Tk sets the prepass argument to postscriptProc to 1 during the first pass; the type manager can use prepass to skip all Postscript generation except for calls to Tk_CanvasPsFont. During the second pass prepass will be 0, so the type manager must generate complete Postscript.

**SCALEPROC**

typePtr->scaleProc is invoked by Tk to rescale a canvas item during the scale widget command. The procedure must match the following prototype:

```c
typedef void Tk_ItemScaleProc(
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    double originX,
    double originY,
    double scaleX,
    double scaleY);
```

The canvas and itemPtr arguments have the usual meaning. originX and originY specify an origin relative to which the item is to be scaled, and scaleX and scaleY give the x and y scale factors. The item should adjust its coordinates so that a point in the item that used to have coordinates x and y will have new coordinates x' and y', where

\[
    x' = originX + scaleX \times (x-originX) \\
    y' = originY + scaleY \times (y-originY)
\]

scaleProc must also update the bounding box in the item's header.
TRANSLATEPROC

typePtr−>translateProc is invoked by Tk to translate a canvas item during the move widget command. The procedure must match the following prototype:

```c
typedef void Tk_ItemTranslateProc(
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    double deltaX,
    double deltaY);
```

The canvas and itemPtr arguments have the usual meaning, and deltaX and deltaY give the amounts that should be added to each x and y coordinate within the item. The type manager should adjust the item’s coordinates and update the bounding box in the item’s header.

INDEXPROC

typePtr−>indexProc is invoked by Tk to translate a string index specification into a numerical index, for example during the index widget command. It is only relevant for item types that support indexable text; typePtr−>indexProc may be specified as NULL for non−textual item types. The procedure must match the following prototype:

```c
typedef int Tk_ItemIndexProc(
    Tcl_Interp *interp,
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    char indexString,
    int *indexPtr);
```

The interp, canvas, and itemPtr arguments all have the usual meaning. indexString contains a textual description of an index, and indexPtr points to an integer value that should be filled in with a numerical index. It is up to the type manager to decide what forms of index are supported (e.g., numbers, insert, sel.first, end, etc.). indexProc should return a Tcl completion code and set interp−>result in the event of an error.

ICURSORPROC

typePtr−>icursorProc is invoked by Tk during the icursor widget command to set the position of the insertion cursor in a textual item. It is only relevant for item types that support an insertion cursor; typePtr−>icursorProc may be specified as NULL for item types that don’t support an insertion cursor. The procedure must match the following prototype:

```c
typedef void Tk_ItemCursorProc(
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    int index);
```

canvas and itemPtr have the usual meanings, and index is an index into the item’s text, as returned by a
previous call to `typePtr->insertProc`. The type manager should position the insertion cursor in the item just before the character given by `index`. Whether or not to actually display the insertion cursor is determined by other information provided by `Tk_CanvasGetTextInfo`.

**SELECTIONPROC**

`typePtr->selectionProc` is invoked by Tk during selection retrievals; it must return part or all of the selected text in the item (if any). It is only relevant for item types that support text; `typePtr->selectionProc` may be specified as NULL for non-textual item types. The procedure must match the following prototype:

```c
typedef int Tk_ItemSelectionProc(
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    int offset,
    char *buffer,
    int maxBytes);
```

`canvas` and `itemPtr` have the usual meanings. `offset` is an offset in bytes into the selection where 0 refers to the first byte of the selection; it identifies the first character that is to be returned in this call. `buffer` points to an area of memory in which to store the requested bytes, and `maxBytes` specifies the maximum number of bytes to return. `selectionProc` should extract up to `maxBytes` characters from the selection and copy them to `maxBytes`; it should return a count of the number of bytes actually copied, which may be less than `maxBytes` if there aren't `offset+maxBytes` bytes in the selection.

**INSERTPROC**

`typePtr->insertProc` is invoked by Tk during the `insert` widget command to insert new text into a canvas item. It is only relevant for item types that support text; `typePtr->insertProc` may be specified as NULL for non-textual item types. The procedure must match the following prototype:

```c
typedef void Tk_ItemInsertProc(
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    int index,
    char *string);
```

`canvas` and `itemPtr` have the usual meanings. `index` is an index into the item's text, as returned by a previous call to `typePtr->insertProc`, and `string` contains new text to insert just before the character given by `index`. The type manager should insert the text and recompute the bounding box in the item's header.

**DCHARSPROC**

`typePtr->dCharsProc` is invoked by Tk during the `dchars` widget command to delete a range of text from a canvas item. It is only relevant for item types that support text; `typePtr->dCharsProc` may be specified as NULL for non-textual item types. The procedure must match the following prototype:

```c
typedef void Tk_ItemDCharsProc(
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    int start,
    int end);
```
canvas and itemPtr have the usual meanings. first and last give the indices of the first and last bytes to be deleted, as returned by previous calls to typePtr->indexProc. The type manager should delete the specified characters and update the bounding box in the item's header.

SEE ALSO

Tk_CanvasPsY, Tk_CanvasTextInfo, Tk_CanvasTkwin

KEYWORDS

canvas, focus, item type, selection, type manager

Copyright ' 1994-1995 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

Tk_CreatePhotoImageFormat − define new file format for photo images

SYNOPSIS

#include <tk.h>
Tk_CreatePhotoImageFormat(formatPtr)

ARGUMENTS

Tk_PhotoImageFormat *formatPtr (in)
Structure that defines the new file format.

DESCRIPTION

Tk_CreatePhotoImageFormat is invoked to define a new file format for image data for use with photo images. The code that implements an image file format is called an image file format handler, or handler for short. The photo image code maintains a list of handlers that can be used to read and write data to or from a file. Some handlers may also support reading image data from a string or converting image data to a string format. The user can specify which handler to use with the −format image configuration option or the −format option to the read and write photo image subcommands.
An image file format handler consists of a collection of procedures plus a Tk_PhotoImageFormat structure, which contains the name of the image file format and pointers to six procedures provided by the handler to deal with files and strings in this format. The Tk_PhotoImageFormat structure contains the following fields:

```c
typedef struct Tk_PhotoImageFormat {
    char *name;
    Tk_ImageFileMatchProc *fileMatchProc;
    Tk_ImageStringMatchProc *stringMatchProc;
    Tk_ImageFileReadProc *fileReadProc;
    Tk_ImageStringReadProc *stringReadProc;
    Tk_ImageFileWriteProc *fileWriteProc;
    Tk_ImageStringWriteProc *stringWriteProc;
} Tk_PhotoImageFormat;
```

The handler need not provide implementations of all six procedures. For example, the procedures that handle string data would not be provided for a format in which the image data are stored in binary, and could therefore contain null characters. If any procedure is not implemented, the corresponding pointer in the Tk_PhotoImageFormat structure should be set to NULL. The handler must provide the fileMatchProc procedure if it provides the fileReadProc procedure, and the stringMatchProc procedure if it provides the stringReadProc procedure.

**PORTABILITY**

In Tk 8.2 and earlier, a different interface was used. Tk 8.3 will still support the old format handlers if the format name is in upper case. If you still want to compile old format handlers with Tk8.3, use the flag -DUSE_OLD_IMAGE. This will restore all function prototypes to match the pre–8.3 situation.

**NAME**

formatPtr->name provides a name for the image type. Once Tk_CreatePhotoImageFormat returns, this name may be used in the -format photo image configuration and subcommand option. The manual page for the photo image (photo(n)) describes how image file formats are chosen based on their names and the value given to the -format option. For new format handlers, the name should be in lower case. Pre–8.3 format handlers are assumed to be in upper case.

**FILEMATCHPROC**

formatPtr->fileMatchProc provides the address of a procedure for Tk to call when it is searching for an image file format handler suitable for reading data in a given file. formatPtr->fileMatchProc must match the following prototype:

```c
typedef int Tk_ImageFileMatchProc(
    Tcl_Channel chan,
    CONST char *fileName,
    Tcl_Obj *format,
    int *widthPtr,
    int *heightPtr,
    Tcl_Interp *interp);
```
The `fileName` argument is the name of the file containing the image data, which is open for reading as `chan`. The `format` argument contains the value given for the `−format` option, or NULL if the option was not specified. If the data in the file appears to be in the format supported by this handler, the `formatPtr−>fileMatchProc` procedure should store the width and height of the image in `*widthPtr` and `*heightPtr` respectively, and return 1. Otherwise it should return 0.

**STRINGMATCHPROC**

`formatPtr−>stringMatchProc` provides the address of a procedure for Tk to call when it is searching for an image file format handler for suitable for reading data from a given string. `formatPtr−>stringMatchProc` must match the following prototype:

```c
typedef int Tk_ImageStringMatchProc(
    Tcl_Obj *data,
    Tcl_Obj *format,
    int *widthPtr,
    int *heightPtr,
    Tcl_Interp *interp);
```

The `data` argument points to the object containing the image data. The `format` argument contains the value given for the `−format` option, or NULL if the option was not specified. If the data in the string appears to be in the format supported by this handler, the `formatPtr−>stringMatchProc` procedure should store the width and height of the image in `*widthPtr` and `*heightPtr` respectively, and return 1. Otherwise it should return 0.

**FILEReadPROC**

`formatPtr−>fileReadProc` provides the address of a procedure for Tk to call to read data from an image file into a photo image. `formatPtr−>fileReadProc` must match the following prototype:

```c
typedef int Tk_ImageFileReadProc(
    Tcl_Interp *interp,
    Tcl_Channel chan,
    const char *fileName,
    Tcl_Obj *format,
    PhotoHandle imageHandle,
    int destX, int destY,
    int width, int height,
    int srcX, int srcY);
```

The `interp` argument is the interpreter in which the command was invoked to read the image; it should be used for reporting errors. The image data is in the file named `fileName`, which is open for reading as `chan`. The `format` argument contains the value given for the `−format` option, or NULL if the option was not specified. The image data in the file, or a subimage of it, is to be read into the photo image identified by the handle `imageHandle`. The subimage of the data in the file is of dimensions `width x height` and has its top–left corner at coordinates `(srcX,srcY)`. It is to be stored in the photo image with its top–left corner at coordinates `(destX,destY)` using the `Tk_PhotoPutBlock` procedure. The return value is a standard Tcl return value.
**STRINGREADPROC**

`formatPtr->stringReadProc` provides the address of a procedure for Tk to call to read data from a string into a photo image. `formatPtr->stringReadProc` must match the following prototype:

```c
typedef int Tk_ImagestringReadProc(
    Tcl_Interp * interp,
    Tcl_Obj * data,
    Tcl_Obj * format,
    PhotoHandle imageHandle,
    destX, destY,
    width, height,
    srcX, srcY);
```

The `interp` argument is the interpreter in which the command was invoked to read the image; it should be used for reporting errors. The `data` argument points to the image data in object form. The `format` argument contains the value given for the `-format` option, or NULL if the option was not specified. The image data in the string, or a subimage of it, is to be read into the photo image identified by the handle `imageHandle`. The subimage of the data in the string is of dimensions `width x height` and has its top–left corner at coordinates `(srcX,srcY)`. It is to be stored in the photo image with its top–left corner at coordinates `(destX,destY)` using the `Tk_PhotoPutBlock` procedure. The return value is a standard Tcl return value.

**FILEWRITEPROC**

`formatPtr->fileWriteProc` provides the address of a procedure for Tk to call to write data from a photo image to a file. `formatPtr->fileWriteProc` must match the following prototype:

```c
typedef int Tk_ImagewriterProc(
    Tcl_Interp * interp,
    const char * fileName,
    Tcl_Obj * format,
    Tk_PhotoImageBlock * blockPtr);
```

The `interp` argument is the interpreter in which the command was invoked to write the image; it should be used for reporting errors. The image data to be written are in memory and are described by the `Tk_PhotoImageBlock` structure pointed to by `blockPtr`; see the manual page FindPhoto(3) for details. The `fileName` argument points to the string giving the name of the file in which to write the image data. The `format` argument contains the value given for the `-format` option, or NULL if the option was not specified. The format string can contain extra characters after the name of the format. If appropriate, the `formatPtr->fileWriteProc` procedure may interpret these characters to specify further details about the image file. The return value is a standard Tcl return value.

**STRINGWRITEPROC**

`formatPtr->stringWriteProc` provides the address of a procedure for Tk to call to translate image data from a photo image into a string. `formatPtr->stringWriteProc` must match the following prototype:
typedef int Tk_ImageStringWriteProc(
    Tcl_Interp *interp,
    Tcl_Obj *format,
    Tk_PhotoImageBlock *blockPtr);

The interp argument is the interpreter in which the command was invoked to convert the image; it should be used for reporting errors. The image data to be converted are in memory and are described by the Tk_PhotoImageBlock structure pointed to by blockPtr; see the manual page FindPhoto(3) for details. The data for the string should be put in the interpreter interp result. The format argument contains the value given for the -format option, or NULL if the option was not specified. The format string can contain extra characters after the name of the format. If appropriate, the formatPtr->stringWriteProc procedure may interpret these characters to specify further details about the image file. The return value is a standard Tcl return value.

SEE ALSO

Tk_FindPhoto, Tk_PhotoPutBlock

KEYWORDS

photo image, image file

Copyright ' 1994 The Australian National University
Copyright ' 1994-1997 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
CrtSelHdlr

NAME

Tk_CreateSelHandler, Tk_DeleteSelHandler – arrange to handle requests for a selection

SYNOPSIS

#include <tk.h>

Tk_CreateSelHandler(tkwin, selection, target, proc, clientData, format)
Tk_DeleteSelHandler(tkwin, selection, target)

ARGUMENTS

Tk_Window tkwin (in)
  Window for which proc will provide selection information.
Atom selection (in)
  The name of the selection for which proc will provide selection information.
Atom target (in)
  Form in which proc can provide the selection (e.g. STRING or FILE_NAME). Corresponds to type arguments in selection commands.
Tk_SelectionProc *proc (in)
  Procedure to invoke whenever the selection is owned by tkwin and the selection contents are requested in the format given by target.
ClientData clientData (in)
  Arbitrary one-word value to pass to proc.
Atom format (in)
  If the selection requestor isn’t in this process, format determines the representation used to transmit the selection to its requestor.

DESCRIPTION

Tk_CreateSelHandler arranges for a particular procedure (proc) to be called whenever selection is owned by tkwin and the selection contents are requested in the form given by target. Target should be one of the entries defined in the left column of Table 2 of the X Inter-Client Communication Conventions Manual (ICCCM) or any other form in which an application is willing to present the selection. The most common form is STRING.

Proc should have arguments and result that match the type Tk_SelectionProc:

typedef int Tk_SelectionProc(
    ClientData,  
    offset,  
    buffer,  
    maxBytes);  

The `clientData` parameter to `proc` is a copy of the `clientData` argument given to `Tk_CreateSelHandler`. Typically, `clientData` points to a data structure containing application-specific information that is needed to retrieve the selection. `Offset` specifies an offset position into the selection, `buffer` specifies a location at which to copy information about the selection, and `maxBytes` specifies the amount of space available at `buffer`. `Proc` should place a NULL-terminated string at `buffer` containing `maxBytes` or fewer characters (not including the terminating NULL), and it should return a count of the number of non-NULL characters stored at `buffer`. If the selection no longer exists (e.g. it once existed but the user deleted the range of characters containing it), then `proc` should return −1.

When transferring large selections, Tk will break them up into smaller pieces (typically a few thousand bytes each) for more efficient transmission. It will do this by calling `proc` one or more times, using successively higher values of `offset` to retrieve successive portions of the selection. If `proc` returns a count less than `maxBytes` it means that the entire remainder of the selection has been returned. If `proc`'s return value is `maxBytes` it means there may be additional information in the selection, so Tk must make another call to `proc` to retrieve the next portion.

`Proc` always returns selection information in the form of a character string. However, the ICCCM allows for information to be transmitted from the selection owner to the selection requestor in any of several formats, such as a string, an array of atoms, an array of integers, etc. The `format` argument to `Tk_CreateSelHandler` indicates what format should be used to transmit the selection to its requestor (see the middle column of Table 2 of the ICCCM for examples). If `format` is not `STRING`, then Tk will take the value returned by `proc` and divided it into fields separated by white space. If `format` is `ATOM`, then Tk will return the selection as an array of atoms, with each field in `proc`'s result treated as the name of one atom. For any other value of `format`, Tk will return the selection as an array of 32-bit values where each field of `proc`'s result is treated as a number and translated to a 32-bit value. In any event, the `format` atom is returned to the selection requestor along with the contents of the selection.

If `Tk_CreateSelHandler` is called when there already exists a handler for `selection` and `target` on `tkwin`, then the existing handler is replaced with a new one.

`Tk_DeleteSelHandler` removes the handler given by `tkwin`, `selection`, and `target`, if such a handler exists. If there is no such handler then it has no effect.

**KEYWORDS**

`format, handler, selection, target`
CrtWindow

NAME

Tk_CreateWindow, Tk_CreateWindowFromPath, Tk_DestroyWindow, Tk_MakeWindowExist – create or delete window

SYNOPSIS

#include <tk.h>

Tk_Window Tk_CreateWindow(interp, parent, name, topLevScreen)

Tk_Window Tk_CreateAnonymousWindow(interp, parent, topLevScreen)

Tk_Window Tk_CreateWindowFromPath(interp, tkwin, pathName, topLevScreen)

Tk_DestroyWindow(tkwin)

Tk_MakeWindowExist(tkwin)

ARGUMENTS

Tcl_Interp *interp (out)

Tcl interpreter to use for error reporting. If no error occurs, then *interp isn't modified.

Tk_Window parent (in)

Token for the window that is to serve as the logical parent of the new window.

CONST char *name (in)

Name to use for this window. Must be unique among all children of the same parent.

CONST char *topLevScreen (in)

Has same format as screenName. If NULL, then new window is created as an internal window. If non–NULL, new window is created as a top–level window on screen topLevScreen. If topLevScreen is an empty string (``'') then new window is created as top–level window of parent's screen.

Tk_Window tkwin (in)

Token for window.

CONST char *pathName (in)

Name of new window, specified as path name within application (e.g. .a.b.c).

DESCRIPTION

The procedures Tk_CreateWindow, Tk_CreateAnonymousWindow, and Tk_CreateWindowFromPath are used to create new windows for use in Tk–based applications. Each of the procedures returns a token that can be used to manipulate the window in other calls to the Tk library. If the window couldn't be created successfully, then NULL is returned and interp–>result is modified to hold an error message.
Tk supports two different kinds of windows: internal windows and top-level windows. An internal window is an interior window of a Tk application, such as a scrollbar or menu bar or button. A top-level window is one that is created as a child of a screen's root window, rather than as an interior window, but which is logically part of some existing main window. Examples of top-level windows are pop-up menus and dialog boxes.

New windows may be created by calling `Tk_CreateWindow`. If the `topLevScreen` argument is NULL, then the new window will be an internal window. If `topLevScreen` is non-NULL, then the new window will be a top-level window: `topLevScreen` indicates the name of a screen and the new window will be created as a child of the root window of `topLevScreen`. In either case Tk will consider the new window to be the logical child of `parent`: the new window's path name will reflect this fact, options may be specified for the new window under this assumption, and so on. The only difference is that new X window for a top-level window will not be a child of `parent`'s X window. For example, a pull-down menu's `parent` would be the button-like window used to invoke it, which would in turn be a child of the menu bar window. A dialog box might have the application's main window as its parent.

`Tk_CreateAnonymousWindow` differs from `Tk_CreateWindow` in that it creates an unnamed window. This window will be manipulable only using C interfaces, and will not be visible to Tcl scripts. Both interior windows and top-level windows may be created with `Tk_CreateAnonymousWindow`.

`Tk_CreateWindowFromPath` offers an alternate way of specifying new windows. In `Tk_CreateWindowFromPath` the new window is specified with a token for any window in the target application (`tkwin`), plus a path name for the new window. It produces the same effect as `Tk_CreateWindow` and allows both top-level and internal windows to be created, depending on the value of `topLevScreen`. In calls to `Tk_CreateWindowFromPath`, as in calls to `Tk_CreateWindow`, the parent of the new window must exist at the time of the call, but the new window must not already exist.

The window creation procedures don't actually issue the command to X to create a window. Instead, they create a local data structure associated with the window and defer the creation of the X window. The window will actually be created by the first call to `Tk_MapWindow`. Deferred window creation allows various aspects of the window (such as its size, background color, etc.) to be modified after its creation without incurring any overhead in the X server. When the window is finally mapped all of the window attributes can be set while creating the window.

The value returned by a window-creation procedure is not the X token for the window (it can't be, since X hasn't been asked to create the window yet). Instead, it is a token for Tk's local data structure for the window. Most of the Tk library procedures take Tk_Window tokens, rather than X identifiers. The actual X window identifier can be retrieved from the local data structure using the `Tk_WindowId` macro; see the manual entry for `Tk_WindowId` for details.

`Tk_DestroyWindow` deletes a window and all the data structures associated with it, including any event handlers created with `Tk_CreateEventHandler`. In addition, `Tk_DestroyWindow` will delete any children of `tkwin` recursively (where children are defined in the Tk sense, consisting of all windows that were created with the given window as `parent`). If `tkwin` is an internal window, then event handlers interested in destroy events are invoked immediately. If `tkwin` is a top-level or main window, then the event handlers will be invoked later, after X has seen the request and returned an event for it.
If a window has been created but hasn't been mapped, so no X window exists, it is possible to force the creation of the X window by calling \texttt{Tk\_MakeWindowExist}. This procedure issues the X commands to instantiate the window given by \texttt{tkwin}.

**KEYWORDS**

create, deferred creation, destroy, display, internal window, screen, top–level window, window
DeleteImg

NAME

Tk_DeleteImage – Destroy an image.

SYNOPSIS

#include <tk.h>
Tk_DeleteImage(interp, name)

ARGUMENTS

Tcl_Interp *interp (in)
   Interpreter for which the image was created.
CONST char *name (in)
   Name of the image.

DESCRIPTION

Tk_DeleteImage deletes the image given by interp and name, if there is one. All instances of that image will redisplay as empty regions. If the given image does not exist then the procedure has no effect.

KEYWORDS

dele image, image manager

Copyright ' 1995-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
DrawFocHlt

NAME

Tk_DrawFocusHighlight – draw the traversal highlight ring for a widget

SYNOPSIS

#include <tk.h>
Tk_DrawFocusHighlight(tkwin, gc, width, drawable)

ARGUMENTS

Tk_Window tkwin (in)
    Window for which the highlight is being drawn. Used to retrieve the window's dimensions, among
    other things.
GC gc (in)
    Graphics context to use for drawing the highlight.
int width (in)
    Width of the highlight ring, in pixels.
Drawable drawable (in)
    Drawable in which to draw the highlight; usually an offscreen pixmap for double buffering.

DESCRIPTION

Tk_DrawFocusHighlight is a utility procedure that draws the traversal highlight ring for a widget. It is
typically invoked by widgets during redisplay.

KEYWORDS

focus, traversal highlight
FindPhoto

NAME

Tk_FindPhoto, Tk_PhotoPutBlock, Tk_PhotoPutZoomedBlock, Tk_PhotoGetImage, Tk_PhotoBlank,
Tk_PhotoExpand, Tk_PhotoGetSize, Tk_PhotoSetSize – manipulate the image data stored in a photo image.

SYNOPSIS

#include <tk.h>
#include <tkPhoto.h>
Tk_PhotoHandle
Tk_FindPhoto(interp, imageName)
void
Tk_PhotoPutBlock(handle, blockPtr, x, y, width, height, compRule)
void
Tk_PhotoPutZoomedBlock(handle, blockPtr, x, y, width, height,
zoomX, zoomY, subsampleX, subsampleY, compRule)
int
Tk_PhotoGetImage(handle, blockPtr)
void
Tk_PhotoBlank(handle)
void
Tk_PhotoExpand(handle, width, height)
void
Tk_PhotoGetSize(handle, widthPtr, heightPtr)
void
Tk_PhotoSetSize(handle, width, height)

ARGUMENTS

Tcl_Interp *interp (in)
Interpreter in which image was created.
CONST char *imageName (in)
Name of the photo image.
Tk_PhotoHandle handle (in)
Opaque handle identifying the photo image to be affected.
Tk_PhotoImageBlock *blockPtr (in)
Specifies the address and storage layout of image data.
int x (in)
Specifies the X coordinate where the top–left corner of the block is to be placed within the image.
int y (in)
Specifies the Y coordinate where the top–left corner of the block is to be placed within the image.
int width (in)
Specifies the width of the image area to be affected (for \texttt{Tk\_PhotoPutBlock}) or the desired image width (for \texttt{Tk\_PhotoExpand} and \texttt{Tk\_PhotoSetSize}).

\textbf{int compRule (in)}

Specifies the compositing rule used when combining transparent pixels in a block of data with a photo image. Must be one of \texttt{TK\_PHOTO\_COMPOSITE\_OVERLAY} (which puts the block of data over the top of the existing photo image, with the previous contents showing through in the transparent bits) or \texttt{TK\_PHOTO\_COMPOSITE\_SET} (which discards the existing photo image contents in the rectangle covered by the data block.)

\textbf{int height (in)}

Specifies the height of the image area to be affected (for \texttt{Tk\_PhotoPutBlock}) or the desired image height (for \texttt{Tk\_PhotoExpand} and \texttt{Tk\_PhotoSetSize}).

\textbf{int \*widthPtr (out)}

Pointer to location in which to store the image width.

\textbf{int \*heightPtr (out)}

Pointer to location in which to store the image height.

\textbf{int subsampleX (in)}

Specifies the subsampling factor in the X direction for input image data.

\textbf{int subsampleY (in)}

Specifies the subsampling factor in the Y direction for input image data.

\textbf{int zoomX (in)}

Specifies the zoom factor to be applied in the X direction to pixels being written to the photo image.

\textbf{int zoomY (in)}

Specifies the zoom factor to be applied in the Y direction to pixels being written to the photo image.

\section*{DESCRIPTION}

\textbf{Tk\_FindPhoto} returns an opaque handle that is used to identify a particular photo image to the other procedures. The parameter is the name of the image, that is, the name specified to the \texttt{image create photo} command, or assigned by that command if no name was specified.

\textbf{Tk\_PhotoPutBlock} is used to supply blocks of image data to be displayed. The call affects an area of the image of size \texttt{width x height} pixels, with its top–left corner at coordinates \((x,y)\). All of \texttt{width}, \texttt{height}, \texttt{x}, and \texttt{y} must be non–negative. If part of this area lies outside the current bounds of the image, the image will be expanded to include the area, unless the user has specified an explicit image size with the \texttt{–width} and/or \texttt{–height} widget configuration options (see \texttt{photo(n)}); in that case the area is silently clipped to the image boundaries.

The \textit{block} parameter is a pointer to a \textbf{Tk\_PhotoImageBlock} structure, defined as follows:

\begin{verbatim}
typedef struct {
    unsigned char *pixelPtr;
    int width;
    int height;
    int pitch;
    int pixelSize;
    int offset[4];
} Tk_PhotoImageBlock;
\end{verbatim}
The `pixelPtr` field points to the first pixel, that is, the top–left pixel in the block. The `width` and `height` fields specify the dimensions of the block of pixels. The `pixelSize` field specifies the address difference between two horizontally adjacent pixels. Often it is 3 or 4, but it can have any value. The `pitch` field specifies the address difference between two vertically adjacent pixels. The `offset` array contains the offsets from the address of a pixel to the addresses of the bytes containing the red, green, blue and alpha (transparency) components. These are normally 0, 1, 2 and 3, but can have other values, e.g., for images that are stored as separate red, green and blue planes.

The `compRule` parameter to `Tk_PhotoPutBlock` specifies a compositing rule that says what to do with transparent pixels. The value `TK_PHOTO_COMPOSITE_OVERLAY` says that the previous contents of the photo image should show through, and the value `TK_PHOTO_COMPOSITE_SET` says that the previous contents of the photo image should be completely ignored, and the values from the block be copied directly across. The behavior in Tk8.3 and earlier was equivalent to having `TK_PHOTO_COMPOSITE_OVERLAY` as a compositing rule.

The value given for the `width` and `height` parameters to `Tk_PhotoPutBlock` do not have to correspond to the values specified in `block`. If they are smaller, `Tk_PhotoPutBlock` extracts a sub–block from the image data supplied. If they are larger, the data given are replicated (in a tiled fashion) to fill the specified area. These rules operate independently in the horizontal and vertical directions.

`Tk_PhotoPutZoomedBlock` works like `Tk_PhotoPutBlock` except that the image can be reduced or enlarged for display. The `subsampleX` and `subsampleY` parameters allow the size of the image to be reduced by subsampling. `Tk_PhotoPutZoomedBlock` will use only pixels from the input image whose X coordinates are multiples of `subsampleX`, and whose Y coordinates are multiples of `subsampleY`. For example, an image of 512x512 pixels can be reduced to 256x256 by setting `subsampleX` and `subsampleY` to 2.

The `zoomX` and `zoomY` parameters allow the image to be enlarged by pixel replication. Each pixel of the (possibly subsampled) input image will be written to a block `zoomX` pixels wide and `zoomY` pixels high of the displayed image. Subsampling and zooming can be used together for special effects.

`Tk_PhotoGetImage` can be used to retrieve image data from a photo image. `Tk_PhotoGetImage` fills in the structure pointed to by the `blockPtr` parameter with values that describe the address and layout of the image data that the photo image has stored internally. The values are valid until the image is destroyed or its size is changed. `Tk_PhotoGetImage` returns 1 for compatibility with the corresponding procedure in the old photo widget.

`Tk_PhotoBlank` blanks the entire area of the photo image. Blank areas of a photo image are transparent.

`Tk_PhotoExpand` requests that the widget's image be expanded to be at least `width x height` pixels in size. The width and/or height are unchanged if the user has specified an explicit image width or height with the `−width` and/or `−height` configuration options, respectively. If the image data are being supplied in many small blocks, it is more efficient to use `Tk_PhotoExpand` or `Tk_PhotoSetSize` at the beginning rather than allowing the image to expand in many small increments as image blocks are supplied.

`Tk_PhotoSetSize` specifies the size of the image, as if the user had specified the given `width` and `height`
values to the −width and −height configuration options. A value of zero for width or height does not change the image's width or height, but allows the width or height to be changed by subsequent calls to Tk_PhotoPutBlock, Tk_PhotoPutZoomedBlock or Tk_PhotoExpand.

Tk_PhotoGetSize returns the dimensions of the image in *widthPtr and *heightPtr.

PORTABILITY

In Tk 8.3 and earlier, Tk_PhotoPutBlock and Tk_PhotoPutZoomedBlock had different signatures. If you want to compile code that uses the old interface against 8.4 without updating your code, compile it with the flag −DUSE_COMPOSITELESS_PHOTO_PUT_BLOCK. Code linked using Stubs against older versions of Tk will continue to work.

CREDITS

The code for the photo image type was developed by Paul Mackerras, based on his earlier photo widget code.

KEYWORDS

photo, image

Copyright ' 1994 The Australian National University
Copyright ' 1994−1996 Sun Microsystems, Inc.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
FontId

NAME

Tk_FontId, Tk_GetFontMetrics, Tk_PostscriptFontName – accessor functions for fonts

SYNOPSIS

#include <tk.h>
Font
Tk_FontId(tkfont)
void
Tk_GetFontMetrics(tkfont, fmPtr)
int
Tk_PostscriptFontName(tkfont, dsPtr)

ARGUMENTS

Tk_Font tkfont (in)
Opaque font token being queried. Must have been returned by a previous call to Tk_GetFont.

Tk_FontMetrics *fmPtr (out)
Pointer to structure in which the font metrics for tkfont will be stored.

Tcl_DString *dsPtr (out)
Pointer to an initialized Tcl_DString to which the name of the Postscript font that corresponds to tkfont will be appended.

DESCRIPTION

Given a tkfont, Tk_FontId returns the token that should be selected into an XGCValues structure in order to construct a graphics context that can be used to draw text in the specified font.

Tk_GetFontMetrics computes the ascent, descent, and linespace of the tkfont in pixels and stores those values in the structure pointer to by fmPtr. These values can be used in computations such as to space multiple lines of text, to align the baselines of text in different fonts, and to vertically align text in a given region. See the documentation for the font command for definitions of the terms ascent, descent, and linespace, used in font metrics.

Tk_PostscriptFontName maps a tkfont to the corresponding Postscript font name that should be used when printing. The return value is the size in points of the tkfont and the Postscript font name is appended to dsPtr. DsPtr must refer to an initialized Tcl_DString. Given a "reasonable" Postscript printer, the following screen font families should print correctly:

Any other font families may not print correctly because the computed Postscript font name may be incorrect or not exist on the printer.

**DATA STRUCTURES**

The Tk_FontMetrics data structure is used by Tk_GetFontMetrics to return information about a font and is defined as follows:

```c
typedef struct Tk_FontMetrics {
    int ascent;
    int descent;
    int linespace;
} Tk_FontMetrics;
```

The **linespace** field is the amount in pixels that the tallest letter sticks up above the baseline, plus any extra blank space added by the designer of the font.

The **descent** is the largest amount in pixels that any letter sticks below the baseline, plus any extra blank space added by the designer of the font.

The **linespace** is the sum of the ascent and descent. How far apart two lines of text in the same font should be placed so that none of the characters in one line overlap any of the characters in the other line.

**KEYWORDS**

- font

*Copyright ' 1996 Sun Microsystems, Inc.*
*Copyright ' 1995-1997 Roger E. Critchlow Jr.*
FreeXId

NAME

Tk_FreeXId – make X resource identifier available for reuse

SYNOPSIS

#include <tk.h>
Tk_FreeXId(display, id)

ARGUMENTS

Display *display (in)
   Display for which id was allocated.
XID id (in)
   Identifier of X resource (window, font, pixmap, cursor, graphics context, or colormap) that is no
   longer in use.

DESCRIPTION

The default allocator for resource identifiers provided by Xlib is very simple−minded and does not allow
resource identifiers to be re−used. If a long−running application reaches the end of the resource id space, it
will generate an X protocol error and crash. Tk replaces the default id allocator with its own allocator, which
allows identifiers to be reused. In order for this to work, Tk_FreeXId must be called to tell the allocator about
resources that have been freed. Tk automatically calls Tk_FreeXId whenever it frees a resource, so if you use
procedures like Tk_GetFont, Tk_GetGC, and Tk_GetPixmap then you need not call Tk_FreeXId.
However, if you allocate resources directly from Xlib, for example by calling XCreatePixmap, then you
should call Tk_FreeXId when you call the corresponding Xlib free procedure, such as XFreePixmap. If you
don't call Tk_FreeXId then the resource identifier will be lost, which could cause problems if the application
runs long enough to lose all of the available identifiers.

KEYWORDS

resource identifier

Copyright ' 1990 The Regents of the University of California.
Copyright ' 1994−1996 Sun Microsystems, Inc.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
GeomReq

NAME

Tk_GeometryRequest, Tk_SetMinimumRequestSize, Tk_SetInternalBorder, Tk_SetInternalBorderEx – specify desired geometry or internal border for a window

SYNOPSIS

#include <tk.h>
Tk_GeometryRequest(tkwin, reqWidth, reqHeight)
Tk_SetMinimumRequestSize(tkwin, minWidth, minHeight)
Tk_SetInternalBorder(tkwin, width)
Tk_SetInternalBorderEx(tkwin, left, right, top, bottom)

ARGUMENTS

Tk_Window tkwin (in)
  Window for which geometry is being requested.
int reqWidth (in)
  Desired width for tkwin, in pixel units.
int reqHeight (in)
  Desired height for tkwin, in pixel units.
int minWidth (in)
  Desired minimum requested width for tkwin, in pixel units.
int minHeight (in)
  Desired minimum requested height for tkwin, in pixel units.
int width (in)
  Space to leave for internal border for tkwin, in pixel units.
int left (in)
  Space to leave for left side of internal border for tkwin, in pixel units.
int right (in)
  Space to leave for right side of internal border for tkwin, in pixel units.
int top (in)
  Space to leave for top side of internal border for tkwin, in pixel units.
int bottom (in)
  Space to leave for bottom side of internal border for tkwin, in pixel units.

DESCRIPTION

Tk_GeometryRequest is called by widget code to indicate its preference for the dimensions of a particular window. The arguments to Tk_GeometryRequest are made available to the geometry manager for the window, which then decides on the actual geometry for the window. Although geometry managers generally try to satisfy requests made to Tk_GeometryRequest, there is no guarantee that this will always be possible.
Widget code should not assume that a geometry request will be satisfied until it receives a `ConfigureNotify` event indicating that the geometry change has occurred. Widget code should never call procedures like `Tk_ResizeWindow` directly. Instead, it should invoke `Tk_GeometryRequest` and leave the final geometry decisions to the geometry manager.

If `tkwin` is a top-level window, then the geometry information will be passed to the window manager using the standard ICCCM protocol.

`Tk_SetInternalBorder` is called by widget code to indicate that the widget has an internal border. This means that the widget draws a decorative border inside the window instead of using the standard X borders, which are external to the window's area. For example, internal borders are used to draw 3-D effects. `Width` specifies the width of the border in pixels. Geometry managers will use this information to avoid placing any children of `tkwin` overlapping the outermost `width` pixels of `tkwin`'s area.

`Tk_SetInternalBorderEx` works like `Tk_SetInternalBorder` but lets you specify different widths for different sides of the window.

`Tk_SetMinimumRequestSize` is called by widget code to indicate that a geometry manager should request at least this size for the widget. This allows a widget to have some control over its size when a propagating geometry manager is used inside it.

The information specified in calls to `Tk_GeometryRequest`, `Tk_SetMinimumRequestSize`, `Tk_SetInternalBorder` and `Tk_SetInternalBorderEx` can be retrieved using the macros `Tk_ReqWidth`, `Tk_ReqHeight`, `Tk_MinReqWidth`, `Tk_MinReqHeight`, `Tk_MinReqWidth`, `Tk_InternalBorderLeft`, `Tk_InternalBorderRight`, `Tk_InternalBorderTop` and `Tk_InternalBorderBottom`. See the `Tk_WindowId` manual entry for details.

**KEYWORDS**

genometry, request
GetAnchor

NAME

Tk_GetAnchorFromObj, Tk_GetAnchor, Tk_NameOfAnchor – translate between strings and anchor positions

SYNOPSIS

#include <tk.h>

int Tk_GetAnchorFromObj(interp, objPtr, anchorPtr)

int Tk_GetAnchor(interp, string, anchorPtr)

CONST char * Tk_NameOfAnchor(anchor)

ARGUMENTS

Tcl_Interp *interp (in)
   Interpreter to use for error reporting, or NULL.

Tcl_Obj *objPtr (in/out)
   String value contains name of anchor point: n, ne, e, se, s, sw, w, nw, or center; internal rep will be modified to cache corresponding Tk_Anchor.

CONST char *string (in)
   Same as objPtr except description of anchor point is passed as a string.

int *anchorPtr (out)
   Pointer to location in which to store anchor position corresponding to objPtr or string.

Tk_Anchor anchor (in)
   Anchor position, e.g. TCL_ANCHOR_CENTER.

DESCRIPTION

Tk_GetAnchorFromObj places in *anchorPtr an anchor position (enumerated type Tk_Anchor) corresponding to objPtr's value. The result will be one of TK_ANCHOR_N, TK_ANCHOR_NE, TK_ANCHOR_E, TK_ANCHOR_SE, TK_ANCHOR_S, TK_ANCHOR_SW, TK_ANCHOR_W, TK_ANCHOR_NW, or TK_ANCHOR_CENTER. Anchor positions are typically used for indicating a point on an object that will be used to position the object, e.g. TK_ANCHOR_N means position the top center point of the object at a particular place.

Under normal circumstances the return value is TCL_OK and interp is unused. If string doesn't contain a valid anchor position or an abbreviation of one of these names, TCL_ERROR is returned. *anchorPtr is unmodified, and an error message is stored in interp's result if interp isn't NULL. Tk_GetAnchorFromObj caches information about the return value in objPtr, which speeds up future calls to Tk_GetAnchorFromObj.
with the same `objPtr`.

`Tk_GetAnchor` is identical to `Tk_GetAnchorFromObj` except that the description of the anchor is specified with a string instead of an object. This prevents `Tk_GetAnchor` from caching the return value, so `Tk_GetAnchor` is less efficient than `Tk_GetAnchorFromObj`.

`Tk_NameOfAnchor` is the logical inverse of `Tk_GetAnchor`. Given an anchor position such as `TK_ANCHOR_N` it returns a statically-allocated string corresponding to `anchor`. If `anchor` isn't a legal anchor value, then ``unknown anchor position`` is returned.

**KEYWORDS**

anchor position

---

Copyright ' 1990 The Regents of the University of California.
Copyright ' 1994-1998 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
GetCapStyl

NAME

Tk_GetCapStyle, Tk_NameOfCapStyle – translate between strings and cap styles

SYNOPSIS

#include <tk.h>

int
Tk_GetCapStyle(interp, string, capPtr)
CONST char *
Tk_NameOfCapStyle(cap)

ARGUMENTS

Tcl_Interp *interp (in)
Interpreter to use for error reporting.

CONST char *string (in)
String containing name of cap style: one of ``butt'', ``projecting'', or ``round''.

int *capPtr (out)
Pointer to location in which to store X cap style corresponding to string.

int cap (in)
Cap style: one of CapButt, CapProjecting, or CapRound.

DESCRIPTION

Tk_GetCapStyle places in *capPtr the X cap style corresponding to string. This will be one of the values CapButt, CapProjecting, or CapRound. Cap styles are typically used in X graphics contexts to indicate how the end-points of lines should be capped. See the X documentation for information on what each style implies.

Under normal circumstances the return value is TCL_OK and interp is unused. If string doesn't contain a valid cap style or an abbreviation of one of these names, then an error message is stored in interp−>result, TCL_ERROR is returned, and *capPtr is unmodified.

Tk_NameOfCapStyle is the logical inverse of Tk_GetCapStyle. Given a cap style such as CapButt it returns a statically−allocated string corresponding to cap. If cap isn't a legal cap style, then ``unknown cap style'' is returned.

KEYWORDS

butt, cap style, projecting, round
GetClrmap

NAME

Tk_GetColormap, Tk_FreeColormap – allocate and free colormaps

SYNOPSIS

#include <tk.h>
Colormap
Tk_GetColormap(interp, tkwin, string)
Tk_FreeColormap(display, colormap)

ARGUMENTS

Tcl_Interp *interp (in)
   Interpreter to use for error reporting.
Tk_Window tkwin (in)
   Token for window in which colormap will be used.
CONST char *string (in)
   Selects a colormap: either new or the name of a window with the same screen and visual as tkwin.
Display *display (in)
   Display for which colormap was allocated.
Colormap colormap (in)
   Colormap to free; must have been returned by a previous call to Tk_GetColormap or Tk_GetVisual.

DESCRIPTION

These procedures are used to manage colormaps. Tk_GetColormap returns a colormap suitable for use in tkwin. If its string argument is new then a new colormap is created; otherwise string must be the name of another window with the same screen and visual as tkwin, and the colormap from that window is returned. If string doesn't make sense, or if it refers to a window on a different screen from tkwin or with a different visual than tkwin, then Tk_GetColormap returns None and leaves an error message in interp−>result.

Tk_FreeColormap should be called when a colormap returned by Tk_GetColormap is no longer needed. Tk maintains a reference count for each colormap returned by Tk_GetColormap, so there should eventually be one call to Tk_FreeColormap for each call to Tk_GetColormap. When a colormap's reference count becomes zero, Tk releases the X colormap.

Tk_GetVisual and Tk_GetColormap work together, in that a new colormap created by Tk_GetVisual may later be returned by Tk_GetColormap. The reference counting mechanism for colormaps includes both procedures, so callers of Tk_GetVisual must also call Tk_FreeColormap to release the colormap. If Tk_GetColormap is called with a string value of new then the resulting colormap will never be returned by Tk_GetVisual; however, it can be used in other windows by calling Tk_GetColormap with the original...
window's name as *string*.

**KEYWORDS**

*colormap*
GetDash

NAME

Tk_GetDash − convert from string to valid dash structure.

SYNOPSIS

#include <tk.h>

int Tk_GetDash(interp, string, dashPtr)

ARGUMENTS

Tcl_Interp *interp (in)
Interpreter to use for error reporting.

CONST char *string (in)
Textual value to be converted.

Tk_Dash *dashPtr (out)
Points to place to store the dash pattern value converted from string.

DESCRIPTION

These procedure parses the string and fills in the result in the Tk_Dash structure. The string can be a list of
integers or a character string containing only [.,−_] or spaces. If all goes well, TCL_OK is returned. If string
doesn't have the proper syntax then TCL_ERROR is returned, an error message is left in the interpreter's
result, and nothing is stored at *dashPtr.

The first possible syntax is a list of integers. Each element represents the number of pixels of a line segment.
Only the odd segments are drawn using the "outline" color. The other segments are drawn transparent.

The second possible syntax is a character list containing only 5 possible characters [.,−_]. The space can be
used to enlarge the space between other line elements, and can not occur as the first posibion in the string.
Some examples: −dash . = −dash {2 4} −dash − = −dash {6 4} −dash −. = −dash {6 4 2 4} −dash −.. = −dash
{6 4 2 4 2 4} −dash {. } = −dash {2 8} −dash , = −dash {4 4}

The main difference of this syntax with the previous is that it it shape-conserving. This means that all values
in the dash list will be multiplied by the line width before display. This assures that "." will always be
displayed as a dot and "−" always as a dash regardless of the line width.

On systems where only a limited set of dash patterns, the dash pattern will be displayed as the most close dash
pattern that is available. For example, on Windows only the first 4 of the above examples are available. The
last 2 examples will be displayed identically as the first one.
GetGC

NAME

Tk_GetGC, Tk_FreeGC – maintain database of read–only graphics contexts

SYNOPSIS

#include <tk.h>

GC
Tk_GetGC(tkwin, valueMask, valuePtr)
Tk_FreeGC(display, gc)

ARGUMENTS

Tk_Window tkwin (in)
   Token for window in which the graphics context will be used.
unsigned long valueMask (in)
   Mask of bits (such as GCForeground or GCStipple) indicating which fields of *valuePtr are valid.
XGCValues *valuePtr (in)
   Pointer to structure describing the desired values for the graphics context.
Display *display (in)
   Display for which gc was allocated.
GC gc (in)
   X identifier for graphics context that is no longer needed. Must have been allocated by Tk_GetGC.

DESCRIPTION

Tk_GetGC and Tk_FreeGC manage a collection of graphics contexts being used by an application. The procedures allow graphics contexts to be shared, thereby avoiding the server overhead that would be incurred if a separate GC were created for each use. Tk_GetGC takes arguments describing the desired graphics context and returns an X identifier for a GC that fits the description. The graphics context that is returned will have default values in all of the fields not specified explicitly by valueMask and valuePtr.

Tk_GetGC maintains a database of all the graphics contexts it has created. Whenever possible, a call to Tk_GetGC will return an existing graphics context rather than creating a new one. This approach can substantially reduce server overhead, so Tk_GetGC should generally be used in preference to the Xlib procedure XCreateGC, which creates a new graphics context on each call.

Since the return values of Tk_GetGC are shared, callers should never modify the graphics contexts returned by Tk_GetGC. If a graphics context must be modified dynamically, then it should be created by calling XCreateGC instead of Tk_GetGC.
When a graphics context is no longer needed, `Tk_FreeGC` should be called to release it. There should be exactly one call to `Tk_FreeGC` for each call to `Tk_GetGC`. When a graphics context is no longer in use anywhere (i.e. it has been freed as many times as it has been gotten) `Tk_FreeGC` will release it to the X server and delete it from the database.

**KEYWORDS**

- graphics context
GetHINSTANCE

NAME

Tk_GetHINSTANCE – retrieve the global application instance handle

SYNOPSIS

#include <tk.h>
HINSTANCE
Tk_GetHINSTANCE()

DESCRIPTION

Tk_GetHINSTANCE returns the Windows application instance handle for the Tk application. This function
is only available on Windows platforms.

KEYWORDS

identifier, instance
GetImage

NAME

Tk_GetImage, Tk_RedrawImage, Tk_SizeOfImage, Tk_FreeImage – use an image in a widget

SYNOPSIS

#include <tk.h>
Tk_Image
Tk_GetImage(interp, tkwin, name, changeProc, clientData)
Tk_RedrawImage(image, imageX, imageY, width, height, drawable, drawableX, drawableY)
Tk_SizeOfImage(image, widthPtr, heightPtr)
Tk_FreeImage(image)

ARGUMENTS

Tcl_Interp *interp (in)
   Place to leave error message.
Tk_Window tkwin (in)
   Window in which image will be used.
CONST char *name (in)
   Name of image.
Tk_ImageChangedProc *changeProc (in)
   Procedure for Tk to invoke whenever image content or size changes.
ClientData clientData (in)
   One−word value for Tk to pass to changeProc.
Tk_Image image (in)
   Token for image instance; must have been returned by a previous call to Tk_GetImage.
int imageX (in)
   X−coordinate of upper−left corner of region of image to redisplay (measured in pixels from the image's upper−left corner).
int imageY (in)
   Y−coordinate of upper−left corner of region of image to redisplay (measured in pixels from the image's upper−left corner).
int width (in)
   Width of region of image to redisplay.
int height (in)
   Height of region of image to redisplay.
Drawable drawable (in)
   Where to display image. Must either be window specified to Tk_GetImage or a pixmap compatible with that window.
int drawableX (in)
   Where to display image in drawable: this is the x−coordinate in drawable where x−coordinate
int drawableY (in)
Where to display image in drawable: this is the y-coordinate in drawable where y-coordinate imageY of the image should be displayed.

int widthPtr (out)
Store width of image (in pixels) here.

int heightPtr (out)
Store height of image (in pixels) here.

DESCRIPTION
These procedures are invoked by widgets that wish to display images. Tk_GetImage is invoked by a widget when it first decides to display an image. name gives the name of the desired image and tkwin identifies the window where the image will be displayed. Tk_GetImage looks up the image in the table of existing images and returns a token for a new instance of the image. If the image doesn't exist then Tk_GetImage returns NULL and leaves an error message in interp->result.

When a widget wishes to actually display an image it must call Tk_RedrawImage, identifying the image (image), a region within the image to redisplay (imageX, imageY, width, and height), and a place to display the image (drawable, drawableX, and drawableY). Tk will then invoke the appropriate image manager, which will display the requested portion of the image before returning.

A widget can find out the dimensions of an image by calling Tk_SizeOfImage: the width and height will be stored in the locations given by widthPtr and heightPtr, respectively.

When a widget is finished with an image (e.g., the widget is being deleted or it is going to use a different image instead of the current one), it must call Tk_FreeImage to release the image instance. The widget should never again use the image token after passing it to Tk_FreeImage. There must be exactly one call to Tk_FreeImage for each call to Tk_GetImage.

If the contents or size of an image changes, then any widgets using the image will need to find out about the changes so that they can redisplay themselves. The changeProc and clientData arguments to Tk_GetImage are used for this purpose. changeProc will be called by Tk whenever a change occurs in the image; it must match the following prototype:

typedef void Tk_ImageChangedProc(
    ClientData,
    int x,
    int y,
    int widthPtr,
    int height;
    int imageWidth,
    int imageHeight);

The clientData argument to changeProc is the same as the clientData argument to Tk_GetImage. It is usually a pointer to the widget record for the widget or some other data structure managed by the widget. The arguments x, y, width, and height identify a region within the image that must be redisplayed; they are
specified in pixels measured from the upper–left corner of the image. The arguments \textit{imageWidth} and \textit{imageHeight} give the image's (new) size.

\textbf{SEE ALSO}

\texttt{Tk_CreateImageType}

\textbf{KEYWORDS}

\texttt{images, redisplay}

\texttt{Copyright} ' 1994 The Regents of the University of California.
\texttt{Copyright} ' 1994-1996 Sun Microsystems, Inc.
\texttt{Copyright} ' 1995-1997 Roger E. Critchlow Jr.
GetJoinStl

NAME

Tk_GetJoinStyle, Tk_NameOfJoinStyle – translate between strings and join styles

SYNOPSIS

#include <tk.h>

int
Tk_GetJoinStyle(interp, string, joinPtr)

CONST char *
Tk_NameOfJoinStyle(join)

ARGUMENTS

Tcl_Interp *interp (in)
Interpreter to use for error reporting.

CONST char *string (in)
String containing name of join style: one of ``bevel'', ``miter'', or ``round''.

int *joinPtr (out)
Pointer to location in which to store X join style corresponding to string.

int join (in)
Join style: one of JoinBevel, JoinMiter, JoinRound.

DESCRIPTION

Tk_GetJoinStyle places in *joinPtr the X join style corresponding to string, which will be one of JoinBevel, JoinMiter, or JoinRound. Join styles are typically used in X graphics contexts to indicate how adjacent line segments should be joined together. See the X documentation for information on what each style implies.

Under normal circumstances the return value is TCL_OK and interp is unused. If string doesn't contain a valid join style or an abbreviation of one of these names, then an error message is stored in interp−>result, TCL_ERROR is returned, and *joinPtr is unmodified.

Tk_NameOfJoinStyle is the logical inverse of Tk_GetJoinStyle. Given a join style such as JoinBevel it returns a statically−allocated string corresponding to join. If join isn't a legal join style, then ``unknown join style'' is returned.

KEYWORDS

bevel, join style, miter, round

Copyright ' 1990 The Regents of the University of California.
GetJustify

NAME

Tk_GetJustifyFromObj, Tk_GetJustify, Tk_NameOfJustify – translate between strings and justification styles

SYNOPSIS

ARGUMENTS

DESCRIPTION

TK_JUSTIFY_LEFT
TK_JUSTIFY_RIGHT
TK_JUSTIFY_CENTER

KEYWORDS

NAME

Tk_GetJustifyFromObj, Tk_GetJustify, Tk_NameOfJustify – translate between strings and justification styles

SYNOPSIS

#include <tk.h>

int Tk_GetJustifyFromObj(interp, objPtr, justifyPtr)

int Tk_GetJustify(interp, string, justifyPtr)

CONST char * Tk_NameOfJustify(justify)

ARGUMENTS

Tcl_Interp *interp (in)

Interpreter to use for error reporting, or NULL.

Tcl_Obj *objPtr (in/out)

String value contains name of justification style (left, right, or center). The internal rep will be modified to cache corresponding justify value.

CONST char *string (in)

Same as objPtr except description of justification style is passed as a string.

int *justifyPtr (out)

Pointer to location in which to store justify value corresponding to objPtr or string.

Tk_Justify justify (in)

Justification style (one of the values listed below).
DESCRIPTION

**Tk_GetJustifyFromObj** places in *justifyPtr the justify value corresponding to objPtr's value. This value will be one of the following:

**TK_JUSTIFY_LEFT**
Means that the text on each line should start at the left edge of the line; as a result, the right edges of lines may be ragged.

**TK_JUSTIFY_RIGHT**
Means that the text on each line should end at the right edge of the line; as a result, the left edges of lines may be ragged.

**TK_JUSTIFY_CENTER**
Means that the text on each line should be centered; as a result, both the left and right edges of lines may be ragged.

Under normal circumstances the return value is **TCL_OK** and interp is unused. If objPtr doesn't contain a valid justification style or an abbreviation of one of these names, **TCL_ERROR** is returned, *justifyPtr is unmodified, and an error message is stored in interp's result if interp isn't NULL.** Tk_GetJustifyFromObj caches information about the return value in objPtr, which speeds up future calls to Tk_GetJustifyFromObj with the same objPtr.

**Tk_GetJustify** is identical to Tk_GetJustifyFromObj except that the description of the justification is specified with a string instead of an object. This prevents Tk_GetJustify from caching the return value, so Tk_GetJustify is less efficient than Tk_GetJustifyFromObj.

**Tk_NameOfJustify** is the logical inverse of Tk_GetJustify. Given a justify value it returns a statically-allocated string corresponding to justify. If justify isn't a legal justify value, then ``unknown justification style'' is returned.

KEYWORDS

center, fill, justification, string

Copyright ' 1990-1994 The Regents of the University of California.
Copyright ' 1994-1998 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
Tk_GetOption

NAME

Tk_GetOption – retrieve an option from the option database

SYNOPSIS

#include <tk.h>

Tk_Uid
Tk_GetOption(tkwin, name, class)

ARGUMENTS

Tk_Window tkwin (in)
    Token for window.
CONST char *name (in)
    Name of desired option.
CONST char *class (in)
    Class of desired option. Null means there is no class for this option; do lookup based on name only.

DESCRIPTION

This procedure is invoked to retrieve an option from the database associated with tkwin’s main window. If there is an option for tkwin that matches the given name or class, then it is returned in the form of a Tk_Uid. If multiple options match name and class, then the highest-priority one is returned. If no option matches, then NULL is returned.

Tk_GetOption caches options related to tkwin so that successive calls for the same tkwin will execute much more quickly than successive calls for different windows.

KEYWORDS

class, name, option, retrieve

Copyright ' 1990 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
GetPixels

NAME

Tk_GetPixelsFromObj, Tk_GetPixels, Tk_GetMMFromObj, Tk_GetScreenMM – translate between strings and screen units

SYNOPSIS

ARGUMENTS

DESCRIPTION

<none>

c
i
m
p

KEYWORDS

NAME

Tk_GetPixelsFromObj, Tk_GetPixels, Tk_GetMMFromObj, Tk_GetScreenMM – translate between strings and screen units

SYNOPSIS

#include <tk.h>

int Tk_GetPixelsFromObj(interp, tkwin, objPtr, intPtr)

int Tk_GetPixels(interp, tkwin, string, intPtr)

int Tk_GetMMFromObj(interp, tkwin, objPtr, doublePtr)

int Tk_GetScreenMM(interp, tkwin, string, doublePtr)

ARGUMENTS

Tcl_Interp *interp (in)

Interpreter to use for error reporting.

Tk_Window tkwin (in)

Window whose screen geometry determines the conversion between absolute units and pixels.

Tcl_Obj *objPtr (in/out)

String value specifies a distance on the screen; internal rep will be modified to cache converted distance.

CONST char *string (in)
Same as objPtr except specification of distance is passed as a string.

int *intPtr (out)
  Pointer to location in which to store converted distance in pixels.

double *doublePtr (out)
  Pointer to location in which to store converted distance in millimeters.

DESCRIPTION

These procedures take as argument a specification of distance on the screen (objPtr or string) and compute the corresponding distance either in integer pixels or floating−point millimeters. In either case, objPtr or string specifies a screen distance as a floating−point number followed by one of the following characters that indicates units:

<none>
  The number specifies a distance in pixels.

c
  The number specifies a distance in centimeters on the screen.

i
  The number specifies a distance in inches on the screen.

m
  The number specifies a distance in millimeters on the screen.

p
  The number specifies a distance in printer's points (1/72 inch) on the screen.

Tk_GetPixelsFromObj converts the value of objPtr to the nearest even number of pixels and stores that value at *intPtr. It returns TCL_OK under normal circumstances. If an error occurs (e.g. objPtr contains a number followed by a character that isn't one of the ones above) then TCL_ERROR is returned and an error message is left in interp's result if interp isn't NULL. Tk_GetPixelsFromObj caches information about the return value in objPtr, which speeds up future calls to Tk_GetPixelsFromObj with the same objPtr.

Tk_GetPixels is identical to Tk_GetPixelsFromObj except that the screen distance is specified with a string instead of an object. This prevents Tk_GetPixels from caching the return value, so Tk_GetAnchor is less efficient than Tk_GetPixelsFromObj.

Tk_GetMMFromObj and Tk_GetScreenMM are similar to Tk_GetPixelsFromObj and Tk_GetPixels (respectively) except that they convert the screen distance to millimeters and store a double−precision floating−point result at *doublePtr.

KEYWORDS
centimeters, convert, inches, millimeters, pixels, points, screen units
GetPixmap

NAME

Tk_GetPixmap, Tk_FreePixmap – allocate and free pixmaps

SYNOPSIS

#include <tk.h>
Pixmap
Tk_GetPixmap(display, d, width, height, depth)
Tk_FreePixmap(display, pixmap)

ARGUMENTS

Display *display (in)
    X display for the pixmap.
Drawable d (in)
    Pixmap or window where the new pixmap will be used for drawing.
int width (in)
    Width of pixmap.
int height (in)
    Height of pixmap.
int depth (in)
    Number of bits per pixel in pixmap.
Pixmap pixmap (in)
    Pixmap to destroy.

DESCRIPTION

These procedures are identical to the Xlib procedures XCreatePixmap and XFreePixmap, except that they have extra code to manage X resource identifiers so that identifiers for deleted pixmaps can be reused in the future. It is important for Tk applications to use these procedures rather than XCreatePixmap and XFreePixmap; otherwise long-running applications may run out of resource identifiers.

Tk_GetPixmap creates a pixmap suitable for drawing in d, with dimensions given by width, height, and depth, and returns its identifier. Tk_FreePixmap destroys the pixmap given by pixmap and makes its resource identifier available for reuse.

KEYWORDS

Pixmap, resource identifier

Copyright ' 1990 The Regents of the University of California.
GetRelief

NAME

Tk_GetReliefFromObj, Tk_GetRelief, Tk_NameOfRelief – translate between strings and relief values

SYNOPSIS

#include <tk.h>

int Tk_GetReliefFromObj(interp, objPtr, reliefPtr)
int Tk_GetRelief(interp, name, reliefPtr)
CONST char *Tk_NameOfRelief(relief)

ARGUMENTS

*Tcl_Interp *interp (in)
  Interpreter to use for error reporting.
*Tcl_Obj *objPtr (in/out)
  String value contains name of relief (one of flat, groove, raised, ridge, solid, or sunken); internal rep will be modified to cache corresponding relief value.
char *string (in)
  Same as objPtr except description of relief is passed as a string.
int *reliefPtr (out)
  Pointer to location in which to store relief value corresponding to objPtr or name.
CONST char *name ()
  Name of the relief.
int relief (in)
  Relief value (one of TK_RELIEF_FLAT, TK_RELIEF_RAISED, TK_RELIEF_SUNKEN, TK_RELIEF_GROOVE, TK_RELIEF_SOLID, or TK_RELIEF_RIDGE).

DESCRIPTION

Tk_GetReliefFromObj places in *reliefPtr the relief value corresponding to the value of objPtr. This value will be one of TK_RELIEF_FLAT, TK_RELIEF_RAISED, TK_RELIEF_SUNKEN, TK_RELIEF_GROOVE, TK_RELIEF_SOLID, or TK_RELIEF_RIDGE. Under normal circumstances the return value is TCL_OK and interp is unused. If objPtr doesn't contain one of the valid relief names or an abbreviation of one of them, then TCL_ERROR is returned, *reliefPtr is unmodified, and an error message is stored in interp's result if interp isn't NULL. Tk_GetReliefFromObj caches information about the return value in objPtr, which speeds up future calls to Tk_GetReliefFromObj with the same objPtr.
Tk_GetRelief is identical to Tk_GetReliefFromObj except that the description of the relief is specified with a string instead of an object. This prevents Tk_GetRelief from caching the return value, so Tk_GetRelief is less efficient than Tk_GetReliefFromObj.

Tk_NameOfRelief is the logical inverse of Tk_GetRelief. Given a relief value it returns the corresponding string (flat, raised, sunken, groove, solid, or ridge). If relief isn't a legal relief value, then "unknown relief" is returned.

KEYWORDS

name, relief, string

Copyright ' 1990 The Regents of the University of California.
Copyright ' 1994-1998 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
GetRootCrd

NAME

Tk_GetRootCoords – Compute root–window coordinates of window

SYNOPSIS

#include <tk.h>
Tk_GetRootCoords(tkwin, xPtr, yPtr)

ARGUMENTS

Tk_Window tkwin (in)
   Token for window.
int *xPtr (out)
   Pointer to location in which to store root–window x–coordinate corresponding to left edge of tkwin's border.
int *yPtr (out)
   Pointer to location in which to store root–window y–coordinate corresponding to top edge of tkwin's border.

DESCRIPTION

This procedure scans through the structural information maintained by Tk to compute the root–window coordinates corresponding to the upper–left corner of tkwin's border. If tkwin has no border, then Tk_GetRootCoords returns the root–window coordinates corresponding to location (0,0) in tkwin. Tk_GetRootCoords is relatively efficient, since it doesn't have to communicate with the X server.

KEYWORDS

coordinates, root window

Copyright © 1990 The Regents of the University of California.
Copyright © 1994–1996 Sun Microsystems, Inc.
GetScroll

NAME

Tk_GetScrollInfo, Tk_GetScrollInfoObj – parse arguments for scrolling commands

SYNOPSIS

#include <tk.h>
int Tk_GetScrollInfo(interp, argc, argv, dblPtr, intPtr)
int Tk_GetScrollInfoObj(interp, objc, objv, dblPtr, intPtr)

ARGUMENTS

Tcl_Interp *interp (in)
   Interpreter to use for error reporting.

int argc (in)
   Number of strings in argv array.

CONST char *argv[] (in)
   Argument strings. These represent the entire widget command, of which the first word is typically the
   widget name and the second word is typically xview or yview.

int objc (in)
   Number of Tcl_Obj's in objv array.

Tcl_Obj *CONST objv[] (in)
   Argument objects. These represent the entire widget command, of which the first word is typically the
   widget name and the second word is typically xview or yview.

double *dblPtr (out)
   Filled in with fraction from moveto option, if any.

int *intPtr (out)
   Filled in with line or page count from scroll option, if any. The value may be negative.

DESCRIPTION

Tk_GetScrollInfo parses the arguments expected by widget scrolling commands such as xview and yview. It
receives the entire list of words that make up a widget command and parses the words starting with argv[2].
The words starting with argv[2] must have one of the following forms:

moveto fraction
scroll number units
scroll number pages

Any of the moveto, scroll, units, and pages keywords may be abbreviated. If argv has the moveto form,
TK_SCROLL_MOVETO is returned as result and *dblPtr is filled in with the fraction argument to the
command, which must be a proper real value. If argv has the scroll form, TK_SCROLL_UNITS or
TK_SCROLL_PAGES is returned and intptr is filled in with the number value, which must be a proper
integer. If an error occurs in parsing the arguments, TK_SCROLL_ERROR is returned and an error message
is left in interp−>result.

Tk_GetScrollInfoObj is identical in function to Tk_GetScrollInfo. However, Tk_GetScrollInfoObj
accepts Tcl_Obj style arguments, making it more appropriate for use with new development.

KEYWORDS

parse, scrollbar, scrolling command, xview, yview

Copyright ' 1994 The Regents of the University of California.
Copyright ' 1994−1996 Sun Microsystems, Inc.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
GetSelect

NAME

Tk_GetSelection – retrieve the contents of a selection

SYNOPSIS

```
#include <tk.h>
int Tk_GetSelection(interp, tkwin, selection, target, proc, clientData)
```

ARGUMENTS

- `Tcl_Interp *interp` (in)
  - Interpreter to use for reporting errors.
- `Tk_Window tkwin` (in)
  - Window on whose behalf to retrieve the selection (determines display from which to retrieve).
- `Atom selection` (in)
  - The name of the selection to be retrieved.
- `Atom target` (in)
  - Form in which to retrieve selection.
- `Tk_GetSelProc *proc` (in)
  - Procedure to invoke to process pieces of the selection as they are retrieved.
- `ClientData clientData` (in)
  - Arbitrary one-word value to pass to `proc`.

DESCRIPTION

**Tk_GetSelection** retrieves the selection specified by the atom `selection` in the format specified by `target`. The selection may actually be retrieved in several pieces; as each piece is retrieved, `proc` is called to process the piece. *Proc* should have arguments and result that match the type `Tk_GetSelProc`:

```c
typedef int Tk_GetSelProc(
    ClientData clientData,
    Tcl_Interp *interp,
    char *portion);
```

The `clientData` and `interp` parameters to `proc` will be copies of the corresponding arguments to **Tk_GetSelection**. *Portion* will be a pointer to a string containing part or all of the selection. For large selections, `proc` will be called several times with successive portions of the selection. The X Inter-Client Communication Conventions Manual allows a selection to be returned in formats other than strings, e.g. as an array of atoms or integers. If this happens, Tk converts the selection back into a string before calling `proc`. If a selection is returned as an array of atoms, Tk converts it to a string containing the atom names separated by white space. For any other format besides string, Tk converts a selection to a string containing hexadecimal
values separated by white space.

**Tk_GetSelection** returns to its caller when the selection has been completely retrieved and processed by `proc`, or when a fatal error has occurred (e.g. the selection owner didn't respond promptly). **Tk_GetSelection** normally returns TCL_OK; if an error occurs, it returns TCL_ERROR and leaves an error message in `interp->result`. `Proc` should also return either TCL_OK or TCL_ERROR. If `proc` encounters an error in dealing with the selection, it should leave an error message in `interp->result` and return TCL_ERROR; this will abort the selection retrieval.

**KEYWORDS**

format, get, selection retrieval

---

© 1990-1994 The Regents of the University of California.
© 1994-1996 Sun Microsystems, Inc.
GetUid

NAME

Tk_GetUid, Tk_Uid – convert from string to unique identifier

SYNOPSIS

#include <tk.h>

Tk_Uid
Tk_GetUid(string)

ARGUMENTS

char *string (in)
String for which the corresponding unique identifier is desired.

DESCRIPTION

Tk_GetUid returns the unique identifier corresponding to string. Unique identifiers are similar to atoms in Lisp, and are used in Tk to speed up comparisons and searches. A unique identifier (type Tk_Uid) is a string pointer and may be used anywhere that a variable of type ``char *'' could be used. However, there is guaranteed to be exactly one unique identifier for any given string value. If Tk_GetUid is called twice, once with string a and once with string b, and if a and b have the same string value (strcmp(a, b) == 0), then Tk_GetUid will return exactly the same Tk_Uid value for each call (Tk_GetUid(a) == Tk_GetUid(b)). This means that variables of type Tk_Uid may be compared directly (x == y) without having to call strcmp. In addition, the return value from Tk_GetUid will have the same string value as its argument (strcmp(Tk_GetUid(a), a) == 0).

KEYWORDS

atom, unique identifier

Copyright ' 1990 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
GetVRoot

NAME

Tk_GetVRootGeometry – Get location and size of virtual root for window

SYNOPSIS

#include <tk.h>
Tk_GetVRootGeometry(tkwin, xPtr, yPtr, widthPtr, heightPtr)

ARGUMENTS

Tk_Window tkwin (in)
  Token for window whose virtual root is to be queried.

int xPtr (out)
  Points to word in which to store x–offset of virtual root.

int yPtr (out)
  Points to word in which to store y–offset of virtual root.

int widthPtr (out)
  Points to word in which to store width of virtual root.

int heightPtr (out)
  Points to word in which to store height of virtual root.

DESCRIPTION

TkGetVRootGeometry returns geometry information about the virtual root window associated with tkwin. The "associated" virtual root is the one in which tkwin's nearest top–level ancestor (or tkwin itself if it is a top–level window) has been reparented by the window manager. This window is identified by a __SWM_ROOT or __WM_ROOT property placed on the top–level window by the window manager. If tkwin is not associated with a virtual root (e.g. because the window manager doesn't use virtual roots) then *xPtr and *yPtr will be set to 0 and *widthPtr and *heightPtr will be set to the dimensions of the screen containing tkwin.

KEYWORDS

greenery, height, location, virtual root, width, window manager

Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
Copyright © 1995-1997 Roger E. Critchlow Jr.
NAME

Tk_GetVisual – translate from string to visual

SYNOPSIS

ARGUMENTS

class depth
default
pathName
number
best
?depth?

DESCRIPTION

Tk_GetVisual takes a string description of a visual and finds a suitable X Visual for use in tkwin, if there is one. It returns a pointer to the X Visual structure for the visual and stores the number of bits per pixel for it at

CREDITS

KEYWORDS
*depthPtr. If string is unrecognizable or if no suitable visual could be found, then NULL is returned and Tk_GetVisual leaves an error message in interp−>result. If colormap is non−NULL then Tk_GetVisual also locates an appropriate colormap for use with the result visual and stores its X identifier at *colormapPtr.

The string argument specifies the desired visual in one of the following ways:

class depth
The string consists of a class name followed by an integer depth, with any amount of white space (including none) in between. class selects what sort of visual is desired and must be one of directcolor, grayscale, greyscale, pseudocolor, staticcolor, staticgray, staticgrey, or truecolor, or a unique abbreviation. depth specifies how many bits per pixel are needed for the visual. If possible, Tk_GetVisual will return a visual with this depth; if there is no visual of the desired depth then Tk_GetVisual looks first for a visual with greater depth, then one with less depth.

default
Use the default visual for tkwin's screen.

pathName
Use the visual for the window given by pathName. pathName must be the name of a window on the same screen as tkwin.

number
Use the visual whose X identifier is number.

best ?depth?
Choose the "best possible" visual, using the following rules, in decreasing order of priority: (a) a visual that has exactly the desired depth is best, followed by a visual with greater depth than requested (but as little extra as possible), followed by a visual with less depth than requested (but as great a depth as possible); (b) if no depth is specified, then the deepest available visual is chosen; (c) pseudocolor is better than truecolor or directcolor, which are better than staticcolor, which is better than staticgray or grayscale; (d) the default visual for the screen is better than any other visual.

CREDITS
The idea for Tk_GetVisual, and the first implementation, came from Paul Mackerras.

KEYWORDS
colormap, screen, visual

Copyright ' 1994 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

Tk_Grab, Tk_Ungrab – manipulate grab state in an application

SYNOPSIS

```
#include <tk.h>

int Tk_Grab(interp, tkwin, grabGlobal)
void Tk_Ungrab(tkwin)
```

ARGUMENTS

```
Tcl_Interp *interp (in)
    Interpreter to use for error reporting
Tk_Window tkwin (in)
    Window on whose behalf the pointer is to be grabbed or released
int grabGlobal (in)
    Boolean indicating whether the grab is global or application local
```

DESCRIPTION

These functions are used to set or release a global or application local grab. When a grab is set on a particular window in a Tk application, mouse and keyboard events can only be received by that window and its descendants. Mouse and keyboard events for windows outside the tree rooted at `tkwin` will be redirected to `tkwin`. If the grab is global, then all mouse and keyboard events for windows outside the tree rooted at `tkwin` (even those intended for windows in other applications) will be redirected to `tkwin`. If the grab is application local, only mouse and keyboard events intended for a windows within the same application (but outside the tree rooted at `tkwin`) will be redirected.

`Tk_Grab` sets a grab on a particular window. `Tkwin` specifies the window on whose behalf the pointer is to be grabbed. `GrabGlobal` indicates whether the grab should be global or application local; if it is non-zero, it means the grab should be global. Normally, `Tk_Grab` returns TCL_OK; if an error occurs and the grab cannot be set, TCL_ERROR is returned and an error message is left if `interp`s result. Once this call completes successfully, no window outside the tree rooted at `tkwin` will receive pointer– or keyboard–related events until the next call to `Tk_Ungrab`. If a previous grab was in effect within the application, then it is replaced with a new one.

`Tk_Ungrab` releases a grab on the mouse pointer and keyboard, if there is one set on the window given by `tkwin`. Once a grab is released, pointer and keyboard events will start being delivered to other windows again.
KEYWORDS

grab, window
HWNDToWindow

NAME

Tk(HWNDToWindow) – Find Tk’s window information for a Windows window

SYNOPSIS

#include <tkPlatDecls.h>
Tk_Window
Tk(HWNDToWindow)(hwnd)

ARGUMENTS

HWND hwnd (in)
Windows handle for the window.

DESCRIPTION

Given a Windows HWND window identifier, this procedure returns the corresponding Tk_Window handle. If there is no Tk_Window corresponding to hwnd then NULL is returned.

KEYWORDS

Windows window id

Copyright © 1995−1997 Roger E. Critchlow Jr.
HandleEvent

NAME

Tk_HandleEvent – invoke event handlers for window system events

SYNOPSIS

#include <tk.h>
Tk_HandleEvent(eventPtr)

ARGUMENTS

XEvent *eventPtr (in)
    Pointer to X event to dispatch to relevant handler(s).

DESCRIPTION

Tk_HandleEvent is a lower−level procedure that deals with window events. It is called by Tcl_ServiceEvent
(and indirectly by Tk_DoOneEvent), and in a few other cases within Tk. It makes callbacks to any window
event handlers (created by calls to Tk_CreateEventHandler) that match eventPtr and then returns. In some
cases it may be useful for an application to bypass the Tk event queue and call Tk_HandleEvent directly
instead of calling Tcl_QueueEvent followed by Tcl_ServiceEvent.

This procedure may be invoked recursively. For example, it is possible to invoke Tk_HandleEvent
recursively from a handler called by Tk_HandleEvent. This sort of operation is useful in some modal
situations, such as when a notifier has been popped up and an application wishes to wait for the user to click a
button in the notifier before doing anything else.

KEYWORDS

callback, event, handler, window

Copyright ' 1990−1992 The Regents of the University of California.
Copyright ' 1994−1996 Sun Microsystems, Inc.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
**IdToWindow**

**NAME**

Tk_IdToWindow – Find Tk's window information for an X window

**SYNOPSIS**

```c
#include <tk.h>
Tk_Window Tk_IdToWindow(display, window)
```

**ARGUMENTS**

- `Display *display (in)`
  - X display containing the window.
- `Window window (in)`
  - X id for window.

**DESCRIPTION**

Given an X window identifier and the X display it corresponds to, this procedure returns the corresponding Tk_Window handle. If there is no Tk_Window corresponding to `window` then NULL is returned.

**KEYWORDS**

- X window id

*Copyright ' 1995-1996 Sun Microsystems, Inc.*
*Copyright ' 1995-1997 Roger E. Critchlow Jr.*
**ImgChanged**

**NAME**

Tk_ImageChanged – notify widgets that image needs to be redrawn

**SYNOPSIS**

```c
#include <tk.h>
Tk_ImageChanged(imageMaster, x, y, width, height, imageWidth, imageHeight)
```

**ARGUMENTS**

*Tk_ImageMaster imageMaster (in)*
- Token for image, which was passed to image's `createProc` when the image was created.

*int x (in)*
- X-coordinate of upper-left corner of region that needs redisplay (measured from upper-left corner of image).

*int y (in)*
- Y-coordinate of upper-left corner of region that needs redisplay (measured from upper-left corner of image).

*int width (in)*
- Width of region that needs to be redrawn, in pixels.

*int height (in)*
- Height of region that needs to be redrawn, in pixels.

*int imageWidth (in)*
- Current width of image, in pixels.

*int imageHeight (in)*
- Current height of image, in pixels.

**DESCRIPTION**

An image manager calls `Tk_ImageChanged` for an image whenever anything happens that requires the image to be redrawn. As a result of calling `Tk_ImageChanged`, any widgets using the image are notified so that they can redisplay themselves appropriately. The `imageMaster` argument identifies the image, and `x`, `y`, `width`, and `height` specify a rectangular region within the image that needs to be redrawn. `imageWidth` and `imageHeight` specify the image's (new) size.

An image manager should call `Tk_ImageChanged` during its `createProc` to specify the image's initial size and to force redisplay if there are existing instances for the image. If any of the pixel values in the image should change later on, `Tk_ImageChanged` should be called again with `x`, `y`, `width`, and `height` values that cover all the pixels that changed. If the size of the image should change, then `Tk_ImageChanged` must be called to indicate the new size, even if no pixels need to be redisplayed.
SEE ALSO

Tk_CreateImageType

KEYWORDS

images, redisplay, image size changes

Copyright ' 1994 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
NAME

Tk_Init, Tk_SafeInit – add Tk to an interpreter and make a new Tk application.

SYNOPSIS

ARGUMENTS

DESCRIPTION

Tk_Init is the package initialization procedure for Tk. It is normally invoked by the Tcl_AppInit procedure for an application or by the load command. Tk_Init adds all of Tk’s commands to interp and creates a new Tk application, including its main window. If the initialization is successful Tk_Init returns TCL_OK; if there is an error it returns TCL_ERROR. Tk_Init also leaves a result or error message in interp−>result.
If there is a variable \texttt{argv} in \texttt{interp}, \texttt{Tk_Init} treats the contents of this variable as a list of options for the new Tk application. The options may have any of the forms documented for the \texttt{wish} application (in fact, \texttt{wish} uses \texttt{Tk_Init} to process its command-line arguments).

\textbf{Tk\_SafeInit} is identical to \texttt{Tk\_Init} except that it removes all Tk commands that are considered unsafe. Those commands and the reasons for their exclusion are:

- \textit{bell}  
  Continuous ringing of the bell is a nuisance.

- \textit{clipboard}  
  A malicious script could replace the contents of the clipboard with the string "\texttt{rm -r \*}" and lead to surprises when the contents of the clipboard are pasted.

- \textit{grab}  
  Grab can be used to block the user from using any other applications.

- \textit{menu}  
  Menus can be used to cover the entire screen and to steal input from the user.

- \textit{selection}  
  See \texttt{clipboard}.

- \textit{send}  
  Send can be used to cause unsafe interpreters to execute commands.

- \textit{tk}  
  The \texttt{tk} command recreates the \texttt{send} command, which is unsafe.

- \textit{tkwait}  
  Tkwait can block the containing process forever

- \textit{toplevel}  
  Toplevels can be used to cover the entire screen and to steal input from the user.

- \textit{wm}  
  If toplevels are ever allowed, \texttt{wm} can be used to remove decorations, move windows around, etc.

\textbf{KEYWORDS}

- \texttt{safe}, \texttt{application}, \texttt{initialization}, \texttt{load}, \texttt{main window}

\textit{Copyright} ' 1995-1996 Sun Microsystems, Inc. 
Copyright ' 1995-1997 Roger E. Critchlow Jr.
Tk_InitStubs

NAME

Tk_InitStubs − initialize the Tk stubs mechanism

SYNOPSIS

#include <tk.h>
CONST char *
Tk_InitStubs(interp, version, exact)

ARGUMENTS

Tcl_Interp *interp (in)
Tcl interpreter handle.

char *version (in)
A version string consisting of one or more decimal numbers separated by dots.

int exact (in)
Non−zero means that only the particular Tk version specified by version is acceptable. Zero means
that versions newer than version are also acceptable as long as they have the same major version
number as version.

INTRODUCTION

The Tcl stubs mechanism defines a way to dynamically bind extensions to a particular Tcl implementation at
run time. the stubs mechanism requires no changes to applications incorporating Tcl/Tk interpreters. Only
developers creating C−based Tcl/Tk extensions need to take steps to use the stubs mechanism with their
extensions. See the Tcl_InitStubs page for more information.
Enabling the stubs mechanism for a Tcl/Tk extension requires the following steps:

1) Call `Tcl_InitStubs` in the extension before calling any other Tcl functions.
2) Call `Tk_InitStubs` if the extension before calling any other Tk functions.
3) Define the USE_TCL_STUBS symbol. Typically, you would include the −DUSE_TCL_STUBS flag when compiling the extension.

Link the extension with the Tcl and Tk stubs libraries instead of the standard Tcl and Tk libraries. On Unix platforms, the library names are `libtclstub8.4.a` and `libtkstub8.4.a`; on Windows platforms, the library names are `tclstub84.lib` and `tkstub84.lib` (adjust names with appropriate version number).

**DESCRIPTION**

`Tk_InitStubs` attempts to initialize the Tk stub table pointers and ensure that the correct version of Tk is loaded. In addition to an interpreter handle, it accepts as arguments a version number and a Boolean flag indicating whether the extension requires an exact version match or not. If `exact` is 0, then the extension is indicating that newer versions of Tk are acceptable as long as they have the same major version number as `version`; non–zero means that only the specified `version` is acceptable. `Tcl_InitStubs` returns a string containing the actual version of Tk satisfying the request, or NULL if the Tk version is not acceptable, does not support the stubs mechanism, or any other error condition occurred.

**SEE ALSO**

`Tcl_InitStubs`

**KEYWORDS**

`stubs`
InternAtom

NAME

Tk_InternAtom, Tk_GetAtomName − manage cache of X atoms

SYNOPSIS

#include <tk.h>
Atom
Tk_InternAtom(tkwin, name)
CONST char *
Tk_GetAtomName(tkwin, atom)

ARGUMENTS

Tk_Window tkwin (in)
Token for window. Used to map atom or name relative to a particular display.
CONST char *name (in)
String name for which atom is desired.
Atom atom (in)
Atom for which corresponding string name is desired.

DESCRIPTION

These procedures are similar to the Xlib procedures XInternAtom and XGetAtomName. Tk_InternAtom returns the atom identifier associated with string given by name; the atom identifier is only valid for the display associated with tkwin. Tk_GetAtomName returns the string associated with atom on tkwin's display. The string returned by Tk_GetAtomName is in Tk's storage: the caller need not free this space when finished with the string, and the caller should not modify the contents of the returned string. If there is no atom atom on tkwin's display, then Tk_GetAtomName returns the string ``?bad atom?''.

Tk caches the information returned by Tk_InternAtom and Tk_GetAtomName so that future calls for the same information can be serviced from the cache without contacting the server. Thus Tk_InternAtom and Tk_GetAtomName are generally much faster than their Xlib counterparts, and they should be used in place of the Xlib procedures.

KEYWORDS

atom, cache, display

Copyright ' 1990 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
Tk_Main

NAME

Tk_Main – main program for Tk–based applications

SYNOPSIS

#include <tk.h>
Tk_Main(argc, argv, appInitProc)

ARGUMENTS

int argc (in)
  Number of elements in argv.
char *argv[] (in)
  Array of strings containing command–line arguments.
Tcl_AppInitProc *appInitProc (in)
  Address of an application–specific initialization procedure. The value for this argument is usually
  Tcl_AppInit.

DESCRIPTION

Tk_Main acts as the main program for most Tk–based applications. Starting with Tk 4.0 it is not called main
anymore because it is part of the Tk library and having a function main in a library (particularly a shared
library) causes problems on many systems. Having main in the Tk library would also make it hard to use Tk
in C++ programs, since C++ programs must have special C++ main functions.

Normally each application contains a small main function that does nothing but invoke Tk_Main. Tk_Main
then does all the work of creating and running a wish–like application.

When it is has finished its own initialization, but before it processes commands, Tk_Main calls the procedure
given by the appInitProc argument. This procedure provides a "hook" for the application to perform its own
initialization, such as defining application–specific commands. The procedure must have an interface that
matches the type Tcl_AppInitProc:

typedef int Tcl_AppInitProc(Tcl_Interp *interp);

AppInitProc is almost always a pointer to Tcl_AppInit; for more details on this procedure, see the
documentation for Tcl_AppInit.
MainLoop

NAME

Tk_MainLoop – loop for events until all windows are deleted

SYNOPSIS

#include <tk.h>
Tk_MainLoop()

DESCRIPTION

Tk_MainLoop is a procedure that loops repeatedly calling Tcl_DoOneEvent. It returns only when there are no applications left in this process (i.e. no main windows exist anymore). Most windowing applications will call Tk_MainLoop after initialization; the main execution of the application will consist entirely of callbacks invoked via Tcl_DoOneEvent.

KEYWORDS

application, event, main loop

Copyright ' 1990-1992 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
MainWin

NAME

Tk_MainWindow, Tk_GetNumMainWindows – functions for querying main window information

SYNOPSIS

#include <tk.h>
Tk_Window
Tk_MainWindow(interp)
int
Tk_GetNumMainWindows()

ARGUMENTS

Tcl_Interp *interp (in/out)
Interpreter associated with the application.

DESCRIPTION

A main window is a special kind of toplevel window used as the outermost window in an application.

If interp is associated with a Tk application then Tk_MainWindow returns the application's main window. If there is no Tk application associated with interp then Tk_MainWindow returns NULL and leaves an error message in interp−>result.

Tk_GetNumMainWindows returns a count of the number of main windows currently open in the process.

KEYWORDS

application, main window

Copyright 1990 The Regents of the University of California.
Copyright 1994−1996 Sun Microsystems, Inc.
Copyright 1995−1997 Roger E. Critchlow Jr.
MaintGeom

NAME

Tk_MaintainGeometry, Tk_UnmaintainGeometry – maintain geometry of one window relative to another

SYNOPSIS

#include <tk.h>
Tk_MaintainGeometry(slave, master, x, y, width, height)
Tk_UnmaintainGeometry(slave, master)

ARGUMENTS

Tk_Window slave (in)
    Window whose geometry is to be controlled.
Tk_Window master (in)
    Window relative to which slave's geometry will be controlled.
int x (in)
    Desired x-coordinate of slave in master, measured in pixels from the inside of master's left border to the outside of slave's left border.
int y (in)
    Desired y-coordinate of slave in master, measured in pixels from the inside of master's top border to the outside of slave's top border.
int width (in)
    Desired width for slave, in pixels.
int height (in)
    Desired height for slave, in pixels.

DESCRIPTION

Tk_MaintainGeometry and Tk_UnmaintainGeometry make it easier for geometry managers to deal with slaves whose masters are not their parents. Three problems arise if the master for a slave is not its parent:

[1] The x– and y–position of the slave must be translated from the coordinate system of the master to that of the parent before positioning the slave.

[2] If the master window, or any of its ancestors up to the slave's parent, is moved, then the slave must be repositioned within its parent in order to maintain the correct position relative to the master.

[3] If the master or one of its ancestors is mapped or unmapped, then the slave must be mapped or unmapped to correspond.
None of these problems is an issue if the parent and master are the same. For example, if the master or one of its ancestors is unmapped, the slave is automatically removed by the screen by X.

**Tk_MaintainGeometry** deals with these problems for slaves whose masters aren't their parents, as well as handling the simpler case of slaves whose masters are their parents. **Tk_MaintainGeometry** is typically called by a window manager once it has decided where a slave should be positioned relative to its master. **Tk_MaintainGeometry** translates the coordinates to the coordinate system of slave's parent and then moves and resizes the slave appropriately. Furthermore, it remembers the desired position and creates event handlers to monitor the master and all of its ancestors up to (but not including) the slave's parent. If any of these windows is moved, mapped, or unmapped, the slave will be adjusted so that it is mapped only when the master is mapped and its geometry relative to the master remains as specified by \( x, y, \text{width}, \) and \( \text{height} \).

When a window manager relinquishes control over a window, or if it decides that it does not want the window to appear on the screen under any conditions, it calls **Tk_UnmaintainGeometry**. **Tk_UnmaintainGeometry** unmaps the window and cancels any previous calls to **Tk_MaintainGeometry** for the master–slave pair, so that the slave's geometry and mapped state are no longer maintained automatically. **Tk_UnmaintainGeometry** need not be called by a geometry manager if the slave, the master, or any of the master's ancestors is destroyed: Tk will call it automatically.

If **Tk_MaintainGeometry** is called repeatedly for the same master–slave pair, the information from the most recent call supersedes any older information. If **Tk_UnmaintainGeometry** is called for a master–slave pair that is isn't currently managed, the call has no effect.

**KEYWORDS**

geometry manager, map, master, parent, position, slave, unmap

---

*Copyright ' 1994 The Regents of the University of California.*
*Copyright ' 1994-1996 Sun Microsystems, Inc.*
*Copyright ' 1995-1997 Roger E. Critchlow Jr.*
**ManageGeom**

**NAME**

Tk_ManageGeometry – arrange to handle geometry requests for a window

**SYNOPSIS**

```c
#include <tk.h>
Tk_ManageGeometry(tkwin, mgrPtr, clientData)
```

**ARGUMENTS**

- **Tk_Window tkwin (in)**
  Token for window to be managed.
- **Tk_GeomMgr *mgrPtr (in)**
  Pointer to data structure containing information about the geometry manager, or NULL to indicate that `tkwin`'s geometry shouldn't be managed anymore. The data structure pointed to by `mgrPtr` must be static: Tk keeps a reference to it as long as the window is managed.
- **ClientData clientData (in)**
  Arbitrary one-word value to pass to geometry manager callbacks.

**DESCRIPTION**

`Tk_ManageGeometry` arranges for a particular geometry manager, described by the `mgrPtr` argument, to control the geometry of a particular slave window, given by `tkwin`. If `tkwin` was previously managed by some other geometry manager, the previous manager loses control in favor of the new one. If `mgrPtr` is NULL, geometry management is cancelled for `tkwin`.

The structure pointed to by `mgrPtr` contains information about the geometry manager:

```c
typedef struct {
    char *name;
    Tk_GeomRequestProc *requestProc;
    Tk_GeomLostSlaveProc *lostSlaveProc;
} Tk_GeomMgr;
```

The `name` field is the textual name for the geometry manager, such as `pack` or `place`; this value will be returned by the command `winfo manager`.

`requestProc` is a procedure in the geometry manager that will be invoked whenever `Tk_GeometryRequest` is called by the slave to change its desired geometry. `requestProc` should have arguments and results that match the type `Tk_GeometryRequestProc`:

```c
typedef void Tk_GeometryRequestProc(
```
The parameters to requestProc will be identical to the corresponding parameters passed to Tk_ManageGeometry. clientData usually points to a data structure containing application–specific information about how to manage tkwin's geometry.

The lostSlaveProc field of mgrPtr points to another procedure in the geometry manager. Tk will invoke lostSlaveProc if some other manager calls Tk_ManageGeometry to claim tkwin away from the current geometry manager. lostSlaveProc is not invoked if Tk_ManageGeometry is called with a NULL value for mgrPtr (presumably the current geometry manager has made this call, so it already knows that the window is no longer managed), nor is it called if mgrPtr is the same as the window's current geometry manager. lostSlaveProc should have arguments and results that match the following prototype:

typedef void Tk_GeomLostSlaveProc(
    ClientData clientData,
    Tk_Window tkwin
);

The parameters to lostSlaveProc will be identical to the corresponding parameters passed to Tk_ManageGeometry.

KEYWORDS

callback, geometry, managed, request, unmanaged

Copyright ' 1990−1994 The Regents of the University of California.
Copyright ' 1994−1996 Sun Microsystems, Inc.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
MapWindow

NAME

Tk_MapWindow, Tk_UnmapWindow – map or unmap a window

SYNOPSIS

#include <tk.h>
Tk_Window
Tk_MapWindow(tkwin)
Tk_UnmapWindow(tkwin)

ARGUMENTS

Tk_Window tkwin (in)
    Token for window.

DESCRIPTION

These procedures may be used to map and unmap windows managed by Tk. Tk_MapWindow maps the window given by tkwin, and also creates an X window corresponding to tkwin if it doesn't already exist. See the Tk_CreateWindow manual entry for information on deferred window creation. Tk_UnmapWindow unmaps tkwin's window from the screen.

If tkwin is a child window (i.e. Tk_CreateWindow was used to create a child window), then event handlers interested in map and unmap events are invoked immediately. If tkwin isn't an internal window, then the event handlers will be invoked later, after X has seen the request and returned an event for it.

These procedures should be used in place of the X procedures XMapWindow and XUnmapWindow, since they update Tk's local data structure for tkwin. Applications using Tk should not invoke XMapWindow and XUnmapWindow directly.

KEYWORDS

map, unmap, window

Copyright ' 1990 The Regents of the University of California.
Copyright ' 1994-1997 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
MeasureChar

NAME

Tk_MeasureChars, Tk_TextWidth, Tk_DrawChars, Tk_UnderlineChars – routines to measure and display simple single-line strings.

SYNOPSIS

#include <tk.h>

int Tk_MeasureChars(tkfont, string, numBytes, maxPixels, flags, lengthPtr)

int Tk_TextWidth(tkfont, string, numBytes)

void Tk_DrawChars(display, drawable, gc, tkfont, string, numBytes, x, y)

void Tk_UnderlineChars(display, drawable, gc, tkfont, string, x, y, firstByte, lastByte)

ARGUMENTS

Tk_Font tkfont (in)
Token for font in which text is to be drawn or measured. Must have been returned by a previous call to Tk_GetFont.

const char *string (in)
Text to be measured or displayed. Need not be null terminated. Any non-printing meta-characters in the string (such as tabs, newlines, and other control characters) will be measured or displayed in a platform-dependent manner.

int numBytes (in)
The maximum number of bytes to consider when measuring or drawing string. Must be greater than or equal to 0.

int maxPixels (in)
If maxPixels is >= 0, it specifies the longest permissible line length in pixels. Characters from string are processed only until this many pixels have been covered. If maxPixels is < 0, then the line length is unbounded and the flags argument is ignored.

int flags (in)
Various flag bits OR-ed together: TK_PARTIAL_OK means include a character as long as any part of it fits in the length given by maxPixels; otherwise, a character must fit completely to be considered. TK_WHOLE_WORDS means stop on a word boundary, if possible. If TK_AT_LEAST_ONE is set, it means return at least one character even if no characters could fit in the length given by maxPixels. If TK_AT_LEAST_ONE is set and TK_WHOLE_WORDS is also set, it means that if not even one word fits on the line, return the first few letters of the word that did fit; if not even one letter of the word fit, then the first letter will still be returned.

int *lengthPtr (out)
Filled with the number of pixels occupied by the number of characters returned as the result of
Tk_MeasureChars.

_DISPLAY*display (in)
Display on which to draw.

_Drawable drawable (in)
Window or pixmap in which to draw.

_GC gc (in)
Graphics context for drawing characters. The font selected into this GC must be the same as the
.tkfont.

_int x, y (in)
Coordinates at which to place the left edge of the baseline when displaying _string_.

_int firstByte (in)
The index of the first byte of the first character to underline in the _string_. Underlining begins at the
left edge of this character.

_int lastByte (in)
The index of the first byte of the last character up to which the underline will be drawn. The character
specified by _lastByte will not itself be underlined.

**DESCRIPTION**

These routines are for measuring and displaying simple single–font, single–line, strings. To measure and
display single–font, multi–line, justified text, refer to the documentation for Tk_ComputeTextLayout. There
is no programming interface in the core of Tk that supports multi–font, multi–line text; support for that
behavior must be built on top of simpler layers. Note that the interfaces described here are byte–oriented not
character–oriented, so index values coming from Tcl scripts need to be converted to byte offsets using the
_Tcl_UtfAtIndex and related routines.

A glyph is the displayable picture of a letter, number, or some other symbol. Not all character codes in a given
font have a glyph. Characters such as tabs, newlines/returns, and control characters that have no glyph are
measured and displayed by these procedures in a platform–dependent manner; under X, they are replaced
with backslashed escape sequences, while under Windows and Macintosh hollow or solid boxes may be
substituted. Refer to the documentation for _Tk_ComputeTextLayout for a programming interface that
supports the platform–independent expansion of tab characters into columns and newlines/returns into
multi–line text.

_Tk_MeasureChars is used both to compute the length of a given string and to compute how many characters
from a string fit in a given amount of space. The return value is the number of bytes from _string_ that fit in the
space specified by _maxPixels subject to the conditions described by _flags_. If all characters fit, the return value
will be _numBytes_. _*lengthPtr_ is filled with the computed width, in pixels, of the portion of the string that was
measured. For example, if the return value is 5, then _*lengthPtr_ is filled with the distance between the left
edge of _string_[0] and the right edge of _string_[4].

_Tk_TextWidth is a wrapper function that provides a simpler interface to the _Tk_MeasureChars function.
The return value is how much space in pixels the given _string_ needs.
Tk_DrawChars draws the string at the given location in the given drawable.

Tk_UnderlineChars underlines the given range of characters in the given string. It doesn't draw the characters (which are assumed to have been displayed previously by Tk_DrawChars); it just draws the underline. This procedure is used to underline a few characters without having to construct an underlined font. To produce natively underlined text, the appropriate underlined font should be constructed and used.

KEYWORDS

font

Copyright © 1996 Sun Microsystems, Inc.
Copyright © 1995−1997 Roger E. Critchlow Jr.
MoveToplevel

NAME

Tk_MoveToplevelWindow – Adjust the position of a top–level window

SYNOPSIS

#include <tk.h>
Tk_MoveToplevelWindow(tkwin, x, y)

ARGUMENTS

Tk_Window tkwin (in)
Token for top−level window to move.

int x (in)
New x−coordinate for the top−left pixel of tkwin's border, or the top−left pixel of the decorative border supplied for tkwin by the window manager, if there is one.

int y (in)
New y−coordinate for the top−left pixel of tkwin's border, or the top−left pixel of the decorative border supplied for tkwin by the window manager, if there is one.

DESCRIPTION

In general, a window should never set its own position; this should be done only by the geometry manager that is responsible for the window. For top−level windows the window manager is effectively the geometry manager; Tk provides interface code between the application and the window manager to convey the application's desires to the geometry manager. The desired size for a top–level window is conveyed using the usual Tk_GeometryRequest mechanism. The procedure Tk_MoveToplevelWindow may be used by an application to request a particular position for a top–level window; this procedure is similar in function to the wmn geometry Tcl command except that negative offsets cannot be specified. It is invoked by widgets such as menus that want to appear at a particular place on the screen.

When Tk_MoveToplevelWindow is called it doesn't immediately pass on the new desired location to the window manager; it defers this action until all other outstanding work has been completed, using the Tk_DoWhenIdle mechanism.

KEYWORDS

position, top–level window, window manager

Copyright ' 1990−1993 The Regents of the University of California.
Copyright ' 1994−1996 Sun Microsystems, Inc.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
 Tk_Name, Tk_PathName, Tk_NameToWindow – convert between names and window tokens

SYNOPSIS

#include <tk.h>

Tk_Uid
Tk_Name(tkwin)
char *
Tk_PathName(tkwin)
Tk_Window
Tk_NameToWindow(interp, pathName, tkwin)

ARGUMENTS

Tk_Window tkwin (in)
    Token for window.
Tcl_Interp *interp (out)
    Interpreter to use for error reporting.
CONST char *pathName (in)
    Character string containing path name of window.

DESCRIPTION

Each window managed by Tk has two names, a short name that identifies a window among children of the same parent, and a path name that identifies the window uniquely among all the windows belonging to the same main window. The path name is used more often in Tk than the short name; many commands, like bind, expect path names as arguments.

The Tk_Name macro returns a window’s short name, which is the same as the name argument passed to Tk_CreateWindow when the window was created. The value is returned as a Tk_Uid, which may be used just like a string pointer but also has the properties of a unique identifier (see the manual entry for Tk_GetUid for details).

The Tk_PathName macro returns a hierarchical name for tkwin. Path names have a structure similar to file names in Unix but with dots between elements instead of slashes: the main window for an application has the path name `.`; its children have names like `.`a" and `.`b"; their children have names like `.`a.aa" and `.`b.bb"; and so on. A window is considered to be a child of another window for naming purposes if the second window was named as the first window’s parent when the first window was created. This is not always the same as the X window hierarchy. For example, a pop-up is created as a child of the root window, but its logical parent will usually be a window within the application.
The procedure **Tk_NameToWindow** returns the token for a window given its path name (the *pathName* argument) and another window belonging to the same main window (*tkwin*). It normally returns a token for the named window, but if no such window exists **Tk_NameToWindow** leaves an error message in *interp−>result* and returns NULL. The *tkwin* argument to **Tk_NameToWindow** is needed because path names are only unique within a single application hierarchy. If, for example, a single process has opened two main windows, each will have a separate naming hierarchy and the same path name might appear in each of the hierarchies. Normally *tkwin* is the main window of the desired hierarchy, but this need not be the case: any window in the desired hierarchy may be used.

**KEYWORDS**

name, path name, token, window

---

Copyright ' 1990 The Regents of the University of California.
Copyright ' 1994−1997 Sun Microsystems, Inc.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
**NamaOfImg**

**NAME**

Tk_NameOfImage – Return name of image.

**SYNOPSIS**

```c
#include <tk.h>
CONST char *
Tk_NameOfImage(typePtr)
```

**ARGUMENTS**

*Tk_ImageMaster *masterPtr (in)*

Token for image, which was passed to image manager's `createProc` when the image was created.

**DESCRIPTION**

This procedure is invoked by image managers to find out the name of an image. Given the token for the image, it returns the string name for the image.

**KEYWORDS**

image manager, image name

*Copyright ' 1995-1996 Sun Microsystems, Inc.*

*Copyright ' 1995-1997 Roger E. Critchlow Jr.*
OwnSelect

NAME

Tk_OwnSelection – make a window the owner of the primary selection

SYNOPSIS

#include <tk.h>
Tk_OwnSelection(tkwin, selection, proc, clientData)

ARGUMENTS

Tk_Window tkwin (in)
Window that is to become new selection owner.
Atom selection (in)
The name of the selection to be owned, such as XA_PRIMARY.
Tk_LostSelProc *proc (in)
Procedure to invoke when tkwin loses selection ownership later.
ClientData clientData (in)
Arbitrary one-word value to pass to proc.

DESCRIPTION

Tk_OwnSelection arranges for tkwin to become the new owner of the selection specified by the atom selection. After this call completes, future requests for the selection will be directed to handlers created for tkwin using Tk_CreateSelHandler. When tkwin eventually loses the selection ownership, proc will be invoked so that the window can clean itself up (e.g. by unhighlighting the selection). Proc should have arguments and result that match the type Tk_LostSelProc:

typedef void Tk_LostSelProc(ClientData clientData);

The clientData parameter to proc is a copy of the clientData argument given to Tk_OwnSelection, and is usually a pointer to a data structure containing application-specific information about tkwin.

KEYWORDS

own, selection owner

Copyright ' 1990-1994 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
ParseArgv

NAME

Tk_ParseArgv – process command-line options

SYNOPSIS

ARGUMENTS

DESCRIPTION

TK_ARGV_END
TK_ARGV_CONSTANT
TK_ARGV_INT
TK_ARGV_FLOAT
TK_ARGV_STRING
TK_ARGV_UID
TK_ARGV_CONST_OPTION
TK_ARGV_OPTION_VALUE
TK_ARGV_OPTION_NAME_VALUE
TK_ARGV_HELP
TK_ARGV_REST
TK_ARGV_FUNC
TK_ARGV_GENFUNC

FLAGS

TK_ARGV_DONT_SKIP_FIRST_ARG
TK_ARGV_NO_ABBREV
TK_ARGV_NO_LEFTOVERS
TK_ARGV_NO_DEFAULTS

EXAMPLE

KEYWORDS

NAME

Tk_ParseArgv – process command-line options

SYNOPSIS

#include <tk.h>

int

Tk_ParseArgv(interp, tkwin, argcPtr, argv, argTable, flags)

ARGUMENTS

Tcl_Interp *interp (in)

Interpreter to use for returning error messages.
Tk_Window *tkwin (in)
Window to use when arguments specify Tk options. If NULL, then no Tk options will be processed.

int argcPtr (in/out)
Pointer to number of arguments in argv; gets modified to hold number of unprocessed arguments that
remain after the call.

CONST char **argv (in/out)
Command line arguments passed to main program. Modified to hold unprocessed arguments that
remain after the call.

Tk_ArgvInfo *argTable (in)
Array of argument descriptors, terminated by element with type TK_ARGV_END.

int flags (in)
If non-zero, then it specifies one or more flags that control the parsing of arguments. Different flags
may be OR'ed together. The flags currently defined are TK_ARGV_DONT_SKIP_FIRST_ARG,
TK_ARGV_NO_ABBREV, TK_ARGV_NO_LEFTOVERS, and TK_ARGV_NO_DEFAULTS.

DESCRIPTION

Tk_ParseArgv processes an array of command–line arguments according to a table describing the kinds
of arguments that are expected. Each of the arguments in argv is processed in turn: if it matches one of the
entries in argTable, the argument is processed according to that entry and discarded. The arguments that do
not match anything in argTable are copied down to the beginning of argv (retaining their original order) and
returned to the caller. At the end of the call Tk_ParseArgv sets *argcPtr to hold the number of arguments
that are left in argv, and argv[*argcPtr] will hold the value NULL. Normally, Tk_ParseArgv assumes that
argv[0] is a command name, so it is treated like an argument that doesn't match argTable and returned to the
caller; however, if the TK_ARGV_DONT_SKIP_FIRST_ARG bit is set in flags then argv[0] will be
processed just like the other elements of argv.

Tk_ParseArgv normally returns the value TCL_OK. If an error occurs while parsing the arguments, then
TCL_ERROR is returned and Tk_ParseArgv will leave an error message in interp−>result in the standard
Tcl fashion. In the event of an error return, *argvPtr will not have been modified, but argv could have been
partially modified. The possible causes of errors are explained below.

The argTable array specifies the kinds of arguments that are expected; each of its entries has the following
structure:

typedef struct {
    char *key
    int type
    char *src
    char *dst
    char *help
} Tk_ArgvInfo;

The key field is a string such as ``−display'' or ``−bg'' that is compared with the values in argv. Type indicates
how to process an argument that matches key (more on this below). Src and dst are additional values used in
processing the argument. Their exact usage depends on type, but typically src indicates a value and dst
indicates where to store the value. The char * declarations for src and dst are placeholders: the actual types may be different. Lastly, help is a string giving a brief description of this option; this string is printed when users ask for help about command-line options.

When processing an argument in argv, Tk_ParseArgv compares the argument to each of the key's in argTable. Tk_ParseArgv selects the first specifier whose key matches the argument exactly, if such a specifier exists. Otherwise Tk_ParseArgv selects a specifier for which the argument is a unique abbreviation. If the argument is a unique abbreviation for more than one specifier, then an error is returned. If there is no matching entry in argTable, then the argument is skipped and returned to the caller.

Once a matching argument specifier is found, Tk_ParseArgv processes the argument according to the type field of the specifier. The argument that matched key is called "the matching argument" in the descriptions below. As part of the processing, Tk_ParseArgv may also use the next argument in argv after the matching argument, which is called "the following argument". The legal values for type, and the processing that they cause, are as follows:

**TK_ARGV_END**
Marks the end of the table. The last entry in argTable must have this type; all of its other fields are ignored and it will never match any arguments.

**TK_ARGV_CONSTANT**
Src is treated as an integer and dst is treated as a pointer to an integer. Src is stored at *dst. The matching argument is discarded.

**TK_ARGV_INT**
The following argument must contain an integer string in the format accepted by strtol (e.g. `0` and `0x` prefixes may be used to specify octal or hexadecimal numbers, respectively). Dst is treated as a pointer to an integer; the following argument is converted to an integer value and stored at *dst. Src is ignored. The matching and following arguments are discarded from argv.

**TK_ARGV_FLOAT**
The following argument must contain a floating-point number in the format accepted by strtol. Dst is treated as the address of a double-precision floating point value; the following argument is converted to a double-precision value and stored at *dst. The matching and following arguments are discarded from argv.

**TK_ARGV_STRING**
In this form, dst is treated as a pointer to a (char *); Tk_ParseArgv stores at *dst a pointer to the following argument, and discards the matching argument and the following arguments from argv. Src is ignored.

**TK_ARGV_UID**
This form is similar to TK_ARGV_STRING, except that the argument is turned into a Tk_Uid by calling Tk_GetUid. Dst is treated as a pointer to a Tk_Uid; Tk_ParseArgv stores at *dst the Tk_Uid corresponding to the following argument, and discards the matching and following arguments from argv. Src is ignored.

**TK_ARGV_CONST_OPTION**
This form causes a Tk option to be set (as if the option command had been invoked). The src field is treated as a pointer to a string giving the value of an option, and dst is treated as a pointer to the name of the option. The matching argument is discarded. If tkwin is NULL, then argument specifiers of this type are ignored (as if they did not exist).
**TK_ARGV_OPTION_VALUE**

This form is similar to TK_ARGV_CONST_OPTION, except that the value of the option is taken from the following argument instead of from src. Dst is used as the name of the option. Src is ignored. The matching and following arguments are discarded. If tkwin is NULL, then argument specifiers of this type are ignored (as if they did not exist).

**TK_ARGV_OPTION_NAME_VALUE**

In this case the following argument is taken as the name of a Tk option and the argument after that is taken as the value for that option. Both src and dst are ignored. All three arguments are discarded from argv. If tkwin is NULL, then argument specifiers of this type are ignored (as if they did not exist).

**TK_ARGV_HELP**

When this kind of option is encountered, Tk_ParseArgv uses the help fields of argTable to format a message describing all the valid arguments. The message is placed in interp−>result and Tk_ParseArgv returns TCL_ERROR. When this happens, the caller normally prints the help message and aborts. If the key field of a TK_ARGV_HELP specifier is NULL, then the specifier will never match any arguments; in this case the specifier simply provides extra documentation, which will be included when some other TK_ARGV_HELP entry causes help information to be returned.

**TK_ARGV_REST**

This option is used by programs or commands that allow the last several of their options to be the name and/or options for some other program. If a TK_ARGV_REST argument is found, then Tk_ParseArgv doesn't process any of the remaining arguments; it returns them all at the beginning of argv (along with any other unprocessed arguments). In addition, Tk_ParseArgv treats dst as the address of an integer value, and stores at *dst the index of the first of the TK_ARGV_REST options in the returned argv. This allows the program to distinguish the TK_ARGV_REST options from other unprocessed options that preceded the TK_ARGV_REST.

**TK_ARGV_FUNC**

For this kind of argument, src is treated as the address of a procedure, which is invoked to process the following argument. The procedure should have the following structure:

```c
int
func(dst, key, nextArg)
    dst char *
    key char *
    nextArg char *
{
}
```

The dst and key parameters will contain the corresponding fields from the argTable entry, and nextArg will point to the following argument from argv (or NULL if there aren't any more arguments left in argv). If func uses nextArg (so that Tk_ParseArgv should discard it), then it should return 1. Otherwise it should return 0 and TkParseArgv will process the following argument in the normal fashion. In either event the matching argument is discarded.

**TK_ARGV_GENFUNC**

This form provides a more general procedural escape. It treats src as the address of a procedure, and passes that procedure all of the remaining arguments. The procedure should have the following form:

```c
int
```
genfunc(dst, interp, key, argc, argv)
    dstchar *
    Tcl_Interp *interp;
    keychar *
    argcint
    argv **
{
}

The dst and key parameters will contain the corresponding fields from the argTable entry. Interp will be the same as the interp argument to Tcl_ParseArgv. Arge and argv refer to all of the options after the matching one. Genfunc should behave in a fashion similar to Tk_ParseArgv: parse as many of the remaining arguments as it can, then return any that are left by compacting them to the beginning of argv (starting at argv[0]). Genfunc should return a count of how many arguments are left in argv; Tk_ParseArgv will process them. If genfunc encounters an error then it should leave an error message in interp->result, in the usual Tcl fashion, and return −1; when this happens Tk_ParseArgv will abort its processing and return TCL_ERROR.

FLAGS

TK_ARGV_DONT_SKIP_FIRST_ARG
Tk_ParseArgv normally treats argv[0] as a program or command name, and returns it to the caller just as if it hadn't matched argTable. If this flag is given, then argv[0] is not given special treatment.

TK_ARGV_NO_ABBREV
Normally, Tk_ParseArgv accepts unique abbreviations for key values in argTable. If this flag is given then only exact matches will be acceptable.

TK_ARGV_NO_LEFTOVERS
Normally, Tk_ParseArgv returns unrecognized arguments to the caller. If this bit is set in flags then Tk_ParseArgv will return an error if it encounters any argument that doesn't match argTable. The only exception to this rule is argv[0], which will be returned to the caller with no errors as long as TK_ARGV_DONT_SKIP_FIRST_ARG isn't specified.

TK_ARGV_NO_DEFAULTS
Normally, Tk_ParseArgv searches an internal table of standard argument specifiers in addition to argTable. If this bit is set in flags, then Tk_ParseArgv will use only argTable and not its default table.

EXAMPLE

Here is an example definition of an argTable and some sample command lines that use the options. Note the effect on argc and argv; arguments processed by Tk_ParseArgv are eliminated from argv, and argc is updated to reflect reduced number of arguments.

    /*
     * Define and set default values for globals.
     */
    int debugFlag = 0;
    int numReps = 100;
    char defaultFileName[] = "out";
char *fileName = defaultFileName;
Boolean exec = FALSE;

/*
 * Define option descriptions.
 */
Tk_ArgvInfo argTable[] = {
    {"-X", TK_ARGV_CONSTANT, (char *) 1, (char *) &debugFlag,
     "Turn on debugging printfs"},
    {"-N", TK_ARGV_INT, (char *) NULL, (char *) &numReps,
     "Number of repetitions"},
    {"-of", TK_ARGV_STRING, (char *) NULL, (char *) &fileName,
     "Name of file for output"},
    {"x", TK_ARGV_REST, (char *) NULL, (char *) &exec,
     "File to exec, followed by any arguments (must be last argument)."},
    {(char *) NULL, TK_ARGV_END, (char *) NULL, (char *) NULL,
     (char *) NULL}
};

main(argc, argv)
    int argc;
    char *argv[];
{
    ...

    if (Tk_ParseArgv(interp, tkwin, &argc, argv, argTable, 0) != TCL_OK) {
        fprintf(stderr, "%s\n", interp->result);
        exit(1);
    }

    /*
     * Remainder of the program.
     */
}

Note that default values can be assigned to variables named in argTable: the variables will only be overwritten if the particular arguments are present in argv. Here are some example command lines and their effects.

prog -N 200 infile  # just sets the numReps variable to 200
prog -of out200 infile  # sets fileName to reference "out200"
prog -XN 10 infile  # sets the debug flag, also sets numReps

In all of the above examples, argc will be set by Tk_ParseArgv to 2, argv[0] will be `"prog"`, argv[1] will be `"infile"`, and argv[2] will be NULL.

KEYWORDS

arguments, command line, options
QWinEvent

NAME

Tk_CollapseMotionEvents, Tk_QueueWindowEvent – Add a window event to the Tcl event queue

SYNOPSIS

#include <tk.h>
int
Tk_CollapseMotionEvents(display, collapse)
Tk_QueueWindowEvent(eventPtr, position)

ARGUMENTS

Display *display (in)
   Display for which to control motion event collapsing.
int collapse (in)
   Indicates whether motion events should be collapsed or not.
XEvent *eventPtr (in)
   An event to add to the event queue.
Tcl_QueuePosition position (in)
   Where to add the new event in the queue: TCL_QUEUE_TAIL, TCL_QUEUE_HEAD, or TCL_QUEUE_MARK.

DESCRIPTION

Tk_QueueWindowEvent places a window event on Tcl's internal event queue for eventual servicing. It creates a Tcl_Event structure, copies the event into that structure, and calls Tcl_QueueEvent to add the event to the queue. When the event is eventually removed from the queue it is processed just like all window events.

When multiple motion events are received for the same window in rapid succession, they are collapsed by default. This behavior can be controlled with Tk_CollapseMotionEvents. Tk_CollapseMotionEvents always returns the previous value for collapse behavior on the display.

The position argument to Tk_QueueWindowEvent has the same significance as for Tcl_QueueEvent; see the documentation for Tcl_QueueEvent for details.

KEYWORDS

callback, clock, handler, modal timeout, events

Copyright © 1995–1996 Sun Microsystems, Inc.
Restack

NAME

Tk_RestackWindow – Change a window’s position in the stacking order

SYNOPSIS

#include <tk.h>

int Tk_RestackWindow(tkwin, aboveBelow, other)

ARGUMENTS

Tk_Window tkwin (in)
  Token for window to restack.
int aboveBelow (in)
  Indicates new position of tkwin relative to other; must be Above or Below.
Tk_Window other (in)
  Tkwin will be repositioned just above or below this window. Must be a sibling of tkwin or a
descendant of a sibling. If NULL then tkwin is restacked above or below all siblings.

DESCRIPTION

Tk_RestackWindow changes the stacking order of window relative to its siblings. If other is specified as
NULL then window is repositioned at the top or bottom of its stacking order, depending on whether
aboveBelow is Above or Below. If other has a non–NULL value then window is repositioned just above or
below other.

The aboveBelow argument must have one of the symbolic values Above or Below. Both of these values are
defined by the include file <X11/Xlib.h>.

KEYWORDS

above, below, obscure, stacking order

Copyright ' 1990 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
RestrictEv

NAME

Tk_RestrictEvents – filter and selectively delay X events

SYNOPSIS

#include <tk.h>
Tk_RestrictProc *
Tk_RestrictEvents (proc, clientData, prevClientDataPtr)

ARGUMENTS

Tk_RestrictProc *proc (in)
    Predicate procedure to call to filter incoming X events. NULL means do not restrict events at all.
ClientData clientData (in)
    Arbitrary argument to pass to proc.
ClientData *prevClientDataPtr (out)
    Pointer to place to save argument to previous restrict procedure.

DESCRIPTION

This procedure is useful in certain situations where applications are only prepared to receive certain X events. After Tk_RestrictEvents is called, Tk_DoOneEvent (and hence Tk_MainLoop) will filter X input events through proc. Proc indicates whether a given event is to be processed immediately, deferred until some later time (e.g. when the event restriction is lifted), or discarded. Proc is a procedure with arguments and result that match the type Tk_RestrictProc:

typedef Tk_RestrictAction Tk_RestrictProc(
    ClientData, 
    XEvent *);

The clientData argument is a copy of the clientData passed to Tk_RestrictEvents; it may be used to provide proc with information it needs to filter events. The eventPtr points to an event under consideration. Proc returns a restrict action (enumerated type Tk_RestrictAction) that indicates what Tk_DoOneEvent should do with the event. If the return value is TK_PROCESS_EVENT, then the event will be handled immediately. If the return value is TK_DEFER_EVENT, then the event will be left on the event queue for later processing. If the return value is TK_DISCARD_EVENT, then the event will be removed from the event queue and discarded without being processed.

Tk_RestrictEvents uses its return value and prevClientDataPtr to return information about the current event restriction procedure (a NULL return value means there are currently no restrictions). These values may be used to restore the previous restriction state when there is no longer any need for the current restriction.
There are very few places where `Tk_RestrictEvents` is needed. In most cases, the best way to restrict events is by changing the bindings with the `bind` Tcl command or by calling `Tk_CreateEventHandler` and `Tk_DeleteEventHandler` from C. The main place where `Tk_RestrictEvents` must be used is when performing synchronous actions (for example, if you need to wait for a particular event to occur on a particular window but you don't want to invoke any handlers for any other events). The "obvious" solution in these situations is to call `XNextEvent` or `XWindowEvent`, but these procedures cannot be used because Tk keeps its own event queue that is separate from the X event queue. Instead, call `Tk_RestrictEvents` to set up a filter, then call `Tk_DoOneEvent` to retrieve the desired event(s).

**KEYWORDS**

delay, event, filter, restriction

---

Copyright ' 1990 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
SetAppName

NAME

Tk_SetAppName – Set the name of an application for `send' commands

SYNOPSIS

#include <tk.h>
CONST char *
Tk_SetAppName(tkwin, name)

ARGUMENTS

Tk_Window tkwin (in)
Token for window in application. Used only to select a particular application.

CONST char *name (in)
Name under which to register the application.

DESCRIPTION

Tk_SetAppName associates a name with a given application and records that association on the display containing with the application's main window. After this procedure has been invoked, other applications on the display will be able to use the send command to invoke operations in the application. If name is already in use by some other application on the display, then a new name will be generated by appending ` #2' to name; if this name is also in use, the number will be incremented until an unused name is found. The return value from the procedure is a pointer to the name actually used.

If the application already has a name when Tk_SetAppName is called, then the new name replaces the old name.

Tk_SetAppName also adds a send command to the application's interpreter, which can be used to send commands from this application to others on any of the displays where the application has windows.

The application's name registration persists until the interpreter is deleted or the send command is deleted from interp, at which point the name is automatically unregistered and the application becomes inaccessible via send. The application can be made accessible again by calling Tk_SetAppName.

Tk_SetAppName is called automatically by Tk_Init, so applications don't normally need to call it explicitly.

The command tk appname provides Tcl–level access to the functionality of Tk_SetAppName.
SetCaret

NAME

Tk_SetCaretPos – set the display caret location

SYNOPSIS

#include <tk.h>
int Tk_SetCaretPos(tkwin, x, y, height)

ARGUMENTS

Tk_Window tkwin (in)
    Token for window.
int x (in)
    Window–relative x coordinate.
int y (in)
    Window–relative y coordinate.
int h (in)
    Height of the caret in the window.

DESCRIPTION

Tk_SetCaretPos sets the caret location for the display of the specified Tk_Window tkwin. The caret is the per–display cursor location used for indicating global focus (e.g. to comply with Microsoft Accessibility guidelines), as well as for location of the over–the–spot XIM (X Input Methods) or Windows IME windows.

KEYWORDS

caret, cursor

Copyright © 2002 ActiveState Corporation.
SetClass

NAME

Tk_SetClass, Tk_Class – set or retrieve a window's class

SYNOPSIS

#include <tk.h>

Tk_SetClass(tkwin, class)
Tk_Uid
Tk_Class(tkwin)

ARGUMENTS

Tk_Window tkwin (in)
    Token for window.
char *class (in)
    New class name for window.

DESCRIPTION

Tk_SetClass is called to associate a class with a particular window. The class string identifies the type of the window; all windows with the same general class of behavior (button, menu, etc.) should have the same class. By convention all class names start with a capital letter, and there exists a Tcl command with the same name as each class (except all in lower−case) which can be used to create and manipulate windows of that class. A window's class string is initialized to NULL when the window is created.

For main windows, Tk automatically propagates the name and class to the WM_CLASS property used by window managers. This happens either when a main window is actually created (e.g. in Tk_MakeWindowExist), or when Tk_SetClass is called, whichever occurs later. If a main window has not been assigned a class then Tk will not set the WM_CLASS property for the window.

Tk_Class is a macro that returns the current value of tkwin's class. The value is returned as a Tk_Uid, which may be used just like a string pointer but also has the properties of a unique identifier (see the manual entry for Tk_GetUid for details). If tkwin has not yet been given a class, then Tk_Class will return NULL.

KEYWORDS

class, unique identifier, window, window manager

Copyright ' 1990 The Regents of the University of California.
Copyright ' 1994−1996 Sun Microsystems, Inc.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
SetClassProcs

NAME

Tk_SetClassProcs – register widget specific procedures

SYNOPSIS

#include <tk.h>
Tk_SetClassProcs(tkwin, procs, instanceData)

ARGUMENTS

Tk_Window tkwin (in)
   Token for window to modify.
Tk_ClassProcs *procs (in)
   Pointer to data structure containing widget specific procedures. The data structure pointed to by procs must be static: Tk keeps a reference to it as long as the window exists.
ClientData instanceData (in)
   Arbitrary one-word value to pass to widget callbacks.

DESCRIPTION

Tk_SetClassProcs is called to register a set of procedures that are used as callbacks in different places.

The structure pointed to by procs contains the following:

typedef struct Tk_ClassProcs {
   unsigned int size;
   Tk_ClassWorldChangedProc *worldChangedProc;
   Tk_ClassCreateProc *createProc;
   Tk_ClassModalProc *modalProc;
} Tk_ClassProcs;

The size field is used to simplify future expansion of the structure. It should always be set to (literally) sizeof(Tk_ClassProcs).

worldChangedProc is invoked when the system has altered in some way that requires some reaction from the widget. For example, when a font alias (see the font manual entry) is reconfigured, widgets configured to use that font alias must update their display accordingly. worldChangedProc should have arguments and results that match the type Tk_ClassWorldChangedProc:

typedef void Tk_ClassWorldChangedProc(
   ClientData instanceData);
The *instanceData* parameter passed to the *worldChangedProc* will be identical to the *instanceData* parameter passed to *Tk_SetClassProcs*.

*createProc* is used to create platform–dependant windows. It is invoked by *Tk_MakeWindowExist*. *createProc* should have arguments and results that match the type *Tk_ClassCreateProc*:

```c
typedef Window Tk_ClassCreateProc(
    Tk_Window tkwin,
    Window parent,
    ClientData instanceData);
```

The *tkwin* and *instanceData* parameters will be identical to the *tkwin* and *instanceData* parameters passed to *Tk_SetClassProcs*. The *parent* parameter will be the parent of the window to be created. The *createProc* should return the created window.

*modalProc* is invoked after all bindings on a widget have been triggered in order to handle a modal loop. *modalProc* should have arguments and results that match the type *Tk_ClassModalProc*:

```c
typedef void Tk_ClassModalProc(
    Tk_Window tkwin,
    XEvent *eventPtr);
```

The *tkwin* parameter to *modalProc* will be identical to the *tkwin* parameter passed to *Tk_SetClassProcs*. The *eventPtr* parameter will be a pointer to an XEvent structure describing the event being processed.

**KEYWORDS**

callback, class
SetGrid

NAME

Tk_SetGrid, Tk_UnsetGrid – control the grid for interactive resizing

SYNOPSIS

#include <tk.h>
Tk_SetGrid(tkwin, reqWidth, reqHeight, widthInc, heightInc)
Tk_UnsetGrid(tkwin)

ARGUMENTS

Tk_Window tkwin (in)
    Token for window.
int reqWidth (in)
    Width in grid units that corresponds to the pixel dimension tkwin has requested via Tk_GeometryRequest.
int reqHeight (in)
    Height in grid units that corresponds to the pixel dimension tkwin has requested via Tk_GeometryRequest.
int widthInc (in)
    Width of one grid unit, in pixels.
int heightInc (in)
    Height of one grid unit, in pixels.

DESCRIPTION

Tk_SetGrid turns on gridded geometry management for tkwin's toplevel window and specifies the geometry of the grid. Tk_SetGrid is typically invoked by a widget when its setGrid option is true. It restricts interactive resizing of tkwin's toplevel window so that the space allocated to the toplevel is equal to its requested size plus or minus even multiples of widthInc and heightInc. Furthermore, the reqWidth and reqHeight values are passed to the window manager so that it can report the window's size in grid units during interactive resizes. If tkwin's configuration changes (e.g., the size of a grid unit changes) then the widget should invoke Tk_SetGrid again with the new information.

Tk_UnsetGrid cancels gridded geometry management for tkwin's toplevel window.

For each toplevel window there can be at most one internal window with gridding enabled. If Tk_SetGrid or Tk_UnsetGrid is invoked when some other window is already controlling gridding for tkwin's toplevel, the calls for the new window have no effect.

See the wm manual entry for additional information on gridded geometry management.
KEYWORDS

grid, window, window manager

Copyright ' 1990-1994 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
SetOptions

NAME

Tk_CreateOptionTable, Tk_DeleteOptionTable, Tk_InitOptions, Tk_SetOptions, Tk_FreeSavedOptions, Tk_RestoreSavedOptions, Tk_GetOptionValue, Tk_GetOptionInfo, Tk_FreeConfigOptions, Tk_Offset – process configuration options

SYNOPSIS

ARGUMENTS

DESCRIPTION

TEMPLATES

TK_OPTION_ANCHOR
TK_OPTION_BITMAP
TK_OPTION_BOOLEAN
TK_OPTION_BORDER
TK_OPTION_COLOR
TK_OPTION_CURSOR
TK_OPTION_CUSTOM
TK_OPTION_DOUBLE
TK_OPTION_END
TK_OPTION_FONT
TK_OPTION_INT
TK_OPTION_JUSTIFY
TK_OPTION_PIXELS
TK_OPTION_RELIEF
TK_OPTION_STRING
TK_OPTION_STRING_TABLE
TK_OPTION_SYNONYM
TK_OPTION_WINDOW

STORAGE MANAGEMENT ISSUES

OBJOFFSET VS. INTERNALOFFSET

CUSTOM OPTION TYPES

clientData
interp
Tkwin
valuePtr
recordPtr
internalOffset
saveInternalPtr
flags

KEYWORDS
NAME

Tk_CreateOptionTable, Tk_DeleteOptionTable, Tk_InitOptions, Tk_SetOptions, Tk_FreeSavedOptions, Tk_RestoreSavedOptions, Tk_GetOptionValue, Tk_GetOptionInfo, Tk_FreeConfigOptions, Tk_Offset – process configuration options

SYNOPSIS

#include <tk.h>
Tk_OptionTable
Tk_CreateOptionTable(interp, templatePtr)
Tk_DeleteOptionTable(optionTable)
int
Tk_InitOptions(interp, recordPtr, optionTable, tkwin)
int
Tk_SetOptions(interp, recordPtr, optionTable, objc, objv, tkwin, savePtr, maskPtr)
Tk_FreeSavedOptions(savedPtr)
Tk_RestoreSavedOptions(savedPtr)
Tcl_Obj *
Tk_GetOptionValue(interp, recordPtr, optionTable, namePtr, tkwin)
Tcl_Obj *
Tk_GetOptionInfo(interp, recordPtr, optionTable, namePtr, tkwin)
Tk_FreeConfigOptions(recordPtr, optionTable, tkwin)
int
Tk_Offset(type, field)

ARGUMENTS

Tcl_Interp *interp (in)
A Tcl interpreter. Most procedures use this only for returning error messages; if it is NULL then no error messages are returned. For Tk_CreateOptionTable the value cannot be NULL; it gives the interpreter in which the option table will be used.

Tk_OptionSpec *templatePtr (in)
Points to an array of static information that describes the configuration options that are supported. Used to build a Tk_OptionTable. The information pointed to by this argument must exist for the lifetime of the Tk_OptionTable.

Tk_OptionTable optionTable (in)
Token for an option table. Must have been returned by a previous call to Tk_CreateOptionTable.

char *recordPtr (in/out)
Points to structure in which values of configuration options are stored; fields of this record are modified by procedures such as Tk_SetOptions and read by procedures such as Tk_GetOptionValue.

Tk_Window tkwin (in)
For options such as TK_OPTION_COLOR, this argument indicates the window in which the option will be used. If optionTable uses no window−dependent options, then a NULL value may be supplied.
for this argument.

- **int objc (in)**  
  Number of values in objv.

- **Tcl_Obj *CONST objv[] (in)**  
  Command-line arguments for setting configuring options.

- **Tk_SavedOptions *savePtr (out)**  
  If not NULL, the structure pointed to by this argument is filled in with the old values of any options that were modified and old values are restored automatically if an error occurs in **Tk_SetOptions**.

- **int *maskPtr (out)**  
  If not NULL, the word pointed to by maskPtr is filled in with the bit−wise OR of the typeMask fields for the options that were modified.

- **Tk_SavedOptions *savedPtr (in/out)**  
  Points to a structure previously filled in by **Tk_SetOptions** with old values of modified options.

- **Tcl_Obj *namePtr (in)**  
  The value of this object is the name of a particular option. If NULL is passed to **Tk_GetOptionInfo** then information is returned for all options. Must not be NULL when **Tk_GetOptionValue** is called.

- **type name (in)**  
  The name of the type of a record.

- **field name field (in)**  
  The name of a field in records of type type.

**DESCRIPTION**

These procedures handle most of the details of parsing configuration options such as those for Tk widgets. Given a description of what options are supported, these procedures handle all the details of parsing options and storing their values into a C structure associated with the widget or object. The procedures were designed primarily for widgets in Tk, but they can also be used for other kinds of objects that have configuration options. In the rest of this manual page ``widget'' will be used to refer to the object whose options are being managed; in practice the object may not actually be a widget. The term ``widget record'' is used to refer to the C−level structure in which information about a particular widget or object is stored.

Note: the easiest way to learn how to use these procedures is to look at a working example. In Tk, the simplest example is the code that implements the button family of widgets, which is an **tkButton.c**. Other examples are in **tkSquare.c** and **tkMenu.c**.

In order to use these procedures, the code that implements the widget must contain a static array of Tk_OptionSpec structures. This is a template that describes the various options supported by that class of widget; there is a separate template for each kind of widget. The template contains information such as the name of each option, its type, its default value, and where the value of the option is stored in the widget record. See TEMPLATES below for more detail.

In order to process configuration options efficiently, the static template must be augmented with additional information that is available only at runtime. The procedure **Tk_CreateOptionTable** creates this dynamic information from the template and returns a Tk_OptionTable token that describes both the static and dynamic information. All of the other procedures, such as **Tk_SetOptions**, take a Tk_OptionTable token as argument.
Typically, **Tk_CreateOptionTable** is called the first time that a widget of a particular class is created and the resulting Tk_OptionTable is used in the future for all widgets of that class. A Tk_OptionTable may be used only in a single interpreter, given by the *interp* argument to **Tk_CreateOptionTable**. When an option table is no longer needed **Tk_DeleteOptionTable** should be called to free all of its resources. All of the option tables for a Tcl interpreter are freed automatically if the interpreter is deleted.

**Tk_InitOptions** is invoked when a new widget is created to set the default values for all of the widget's configuration options. **Tk_InitOptions** is passed a token for an option table (*optionTable*) and a pointer to a widget record (*recordPtr*), which is the C structure that holds information about this widget. **Tk_InitOptions** uses the information in the option table to choose an appropriate default for each option, then it stores the default value directly into the widget record, overwriting any information that was already present in the widget record. **Tk_InitOptions** normally returns TCL_OK. If an error occurred while setting the default values (e.g., because a default value was erroneous) then TCL_ERROR is returned and an error message is left in *interp's* result if *interp* isn't NULL.

**Tk_SetOptions** is invoked to modify configuration options based on information specified in a Tcl command. The command might be one that creates a new widget, or a command that modifies options on an existing widget. The *objc* and *objv* arguments describe the values of the arguments from the Tcl command. *Objv* must contain an even number of objects: the first object of each pair gives the name of an option and the second object gives the new value for that option. **Tk_SetOptions** looks up each name in *optionTable*, checks that the new value of the option conforms to the type in *optionTable*, and stores the value of the option into the widget record given by *recordPtr*. **Tk_SetOptions** normally returns TCL_OK. If an error occurred (such as an unknown option name or an illegal option value) then TCL_ERROR is returned and an error message is left in *interp's* result if *interp* isn't NULL.

**Tk_SetOptions** has two additional features. First, if the *maskPtr* argument isn't NULL then it points to an integer value that is filled in with information about the options that were modified. For each option in the template passed to **Tk_CreateOptionTable** there is a *typeMask* field. The bits of this field are defined by the code that implements the widget; for example, each bit might correspond to a particular configuration option. Alternatively, bits might be used functionally. For example, one bit might be used for redisplay: all options that affect the widget's display, such that changing the option requires the widget to be redisplayed, might have that bit set. Another bit might indicate that the geometry of the widget must be recomputed, and so on. **Tk_SetOptions** OR's together the *typeMask* fields from all the options that were modified and returns this value at *maskPtr*; the caller can then use this information to optimize itself so that, for example, it doesn't redisplay the widget if the modified options don't affect the widget's appearance.

The second additional feature of **Tk_SetOptions** has to do with error recovery. If an error occurs while processing configuration options, this feature makes it possible to restore all the configuration options to their previous values. Errors can occur either while processing options in **Tk_SetOptions** or later in the caller. In many cases the caller does additional processing after **Tk_SetOptions** returns; for example, it might use an option value to set a trace on a variable and may detect an error if the variable is an array instead of a scalar. Error recovery is enabled by passing in a non-NULL value for the *savePtr* argument to **Tk_SetOptions**; this should be a pointer to an uninitialized Tk_SavedOptions structure on the caller's stack. **Tk_SetOptions** overwrites the structure pointed to by *savePtr* with information about the old values of any options modified by the procedure. If **Tk_SetOptions** returns successfully, the caller uses the structure in one of two ways. If
the caller completes its processing of the new options without any errors, then it must pass the structure to `Tk_FreeSavedOptions` so that the old values can be freed. If the caller detects an error in its processing of the new options, then it should pass the structure to `Tk_RestoreSavedOptions`, which will copy the old values back into the widget record and free the new values. If `Tk_SetOptions` detects an error then it automatically restores any options that had already been modified and leaves *savePtr in an empty state: the caller need not call either `Tk_FreeSavedOptions` or `Tk_RestoreSavedOptions`. If the savePtr argument to `Tk_SetOptions` is NULL then `Tk_SetOptions` frees each old option value immediately when it sets a new value for the option. In this case, if an error occurs in the third option, the old values for the first two options cannot be restored.

`Tk_GetOptionValue` returns the current value of a configuration option for a particular widget. The namePtr argument contains the name of an option; `Tk_GetOptionValue` uses `optionTable` to lookup the option and extract its value from the widget record pointed to by `recordPtr`, then it returns an object containing that value. If an error occurs (e.g., because namePtr contains an unknown option name) then NULL is returned and an error message is left in interp's result unless interp is NULL.

`Tk_GetOptionInfo` returns information about configuration options in a form suitable for configure widget commands. If the namePtr argument is not NULL, it points to an object that gives the name of a configuration option; `Tk_GetOptionInfo` returns an object containing a list with five elements, which are the name of the option, the name and class used for the option in the option database, the default value for the option, and the current value for the option. If the namePtr argument is NULL, then `Tk_GetOptionInfo` returns information about all options in the form of a list of lists; each sublist describes one option. Synonym options are handled differently depending on whether namePtr is NULL: if namePtr is NULL then the sublist for each synonym option has only two elements, which are the name of the option and the name of the other option that it refers to; if namePtr is non–NULL and names a synonym option then the object returned is the five–element list for the other option that the synonym refers to. If an error occurs (e.g., because namePtr contains an unknown option name) then NULL is returned and an error message is left in interp's result unless interp is NULL.

`Tk_FreeConfigOptions` must be invoked when a widget is deleted. It frees all of the resources associated with any of the configuration options defined in recordPtr by `optionTable`.

The `Tk_Offset` macro is provided as a safe way of generating the objOffset and internalOffset values for entries in Tk_OptionSpec structures. It takes two arguments: the name of a type of record, and the name of a field in that record. It returns the byte offset of the named field in records of the given type.

**TEMPLATES**

The array of Tk_OptionSpec structures passed to `Tk_CreateOptionTable` via its `templatePtr` argument describes the configuration options supported by a particular class of widgets. Each structure specifies one configuration option and has the following fields:

```c
typedef struct {
    Tk_OptionType type;
    char *optionName;
    char *dbName;
    char *dbClass;
} Tk_OptionSpec;
```
char *defValue;
int objOffset;
int internalOffset;
flags
  ClientData;
typeMask;
} Tk_OptionSpec;

The type field indicates what kind of configuration option this is (e.g. TK_OPTION_COLOR for a color value, or TK_OPTION_INT for an integer value). Type determines how the value of the option is parsed (more on this below). The optionName field is a string such as −font or −bg; it is the name used for the option in Tcl commands and passed to procedures via the obj or namePtr arguments. The dbName and dbClass fields are used by Tk_InitOptions to look up a default value for this option in the option database; if dbName is NULL then the option database is not used by Tk_InitOptions for this option. The defValue field specifies a default value for this configuration option if no value is specified in the option database. The objOffset and internalOffset fields indicate where to store the value of this option in widget records (more on this below); values for the objOffset and internalOffset fields should always be generated with the Tk_Offset macro. The flags field contains additional information to control the processing of this configuration option (see below for details). ClientData provides additional type-specific data needed by certain types. For instance, for TK_OPTION_COLOR types, clientData is a string giving the default value to use on monochrome displays. See the descriptions of the different types below for details. The last field, typeMask, is used by Tk_SetOptions to return information about which options were modified; see the description of Tk_SetOptions above for details.

When Tk_InitOptions and Tk_SetOptions store the value of an option into the widget record, they can do it in either of two ways. If the objOffset field of the Tk_OptionSpec is greater than or equal to zero, then the value of the option is stored as a (Tcl_Obj *) at the location in the widget record given by objOffset. If the internalOffset field of the Tk_OptionSpec is greater than or equal to zero, then the value of the option is stored in a type-specific internal form at the location in the widget record given by internalOffset. For example, if the option's type is TK_OPTION_INT then the internal form is an integer. If the objOffset or internalOffset field is negative then the value is not stored in that form. At least one of the offsets must be greater than or equal to zero.

The flags field consists of one or more bits ORed together. At present only a single flag is supported: TK_OPTION_NULL_OK. If this bit is set for an option then an empty string will be accepted as the value for the option and the resulting internal form will be a NULL pointer, a zero value, or None, depending on the type of the option. If the flag is not set then empty strings will result in errors. TK_OPTION_NULL_OK is typically used to allow a feature to be turned off entirely, e.g. set a cursor value to None so that a window simply inherits its parent's cursor. Not all option types support the TK_OPTION_NULL_OK flag; for those that do, there is an explicit indication of that fact in the descriptions below.

The type field of each Tk_OptionSpec structure determines how to parse the value of that configuration option. The legal value for type, and the corresponding actions, are described below. If the type requires a tkwin value to be passed into procedures like Tk_SetOptions, or if it uses the clientData field of the Tk_OptionSpec, then it is indicated explicitly; if not mentioned, the type requires neither tkwin nor clientData.

The type field indicates what kind of configuration option this is (e.g. TK_OPTION_COLOR for a color value, or TK_OPTION_INT for an integer value). Type determines how the value of the option is parsed (more on this below). The optionName field is a string such as −font or −bg; it is the name used for the option in Tcl commands and passed to procedures via the obj or namePtr arguments. The dbName and dbClass fields are used by Tk_InitOptions to look up a default value for this option in the option database; if dbName is NULL then the option database is not used by Tk_InitOptions for this option. The defValue field specifies a default value for this configuration option if no value is specified in the option database. The objOffset and internalOffset fields indicate where to store the value of this option in widget records (more on this below); values for the objOffset and internalOffset fields should always be generated with the Tk_Offset macro. The flags field contains additional information to control the processing of this configuration option (see below for details). ClientData provides additional type-specific data needed by certain types. For instance, for TK_OPTION_COLOR types, clientData is a string giving the default value to use on monochrome displays. See the descriptions of the different types below for details. The last field, typeMask, is used by Tk_SetOptions to return information about which options were modified; see the description of Tk_SetOptions above for details.

When Tk_InitOptions and Tk_SetOptions store the value of an option into the widget record, they can do it in either of two ways. If the objOffset field of the Tk_OptionSpec is greater than or equal to zero, then the value of the option is stored as a (Tcl_Obj *) at the location in the widget record given by objOffset. If the internalOffset field of the Tk_OptionSpec is greater than or equal to zero, then the value of the option is stored in a type-specific internal form at the location in the widget record given by internalOffset. For example, if the option's type is TK_OPTION_INT then the internal form is an integer. If the objOffset or internalOffset field is negative then the value is not stored in that form. At least one of the offsets must be greater than or equal to zero.

The flags field consists of one or more bits ORed together. At present only a single flag is supported: TK_OPTION_NULL_OK. If this bit is set for an option then an empty string will be accepted as the value for the option and the resulting internal form will be a NULL pointer, a zero value, or None, depending on the type of the option. If the flag is not set then empty strings will result in errors. TK_OPTION_NULL_OK is typically used to allow a feature to be turned off entirely, e.g. set a cursor value to None so that a window simply inherits its parent's cursor. Not all option types support the TK_OPTION_NULL_OK flag; for those that do, there is an explicit indication of that fact in the descriptions below.

The type field of each Tk_OptionSpec structure determines how to parse the value of that configuration option. The legal value for type, and the corresponding actions, are described below. If the type requires a tkwin value to be passed into procedures like Tk_SetOptions, or if it uses the clientData field of the Tk_OptionSpec, then it is indicated explicitly; if not mentioned, the type requires neither tkwin nor clientData.
**TK_OPTION_ANCHOR**
The value must be a standard anchor position such as `ne` or `center`. The internal form is a Tk_Anchor value like the ones returned by `Tk_GetAnchorFromObj`.

**TK_OPTION_BITMAP**
The value must be a standard Tk bitmap name. The internal form is a Pixmap token like the ones returned by `Tk_AllocBitmapFromObj`. This option type requires `tkwin` to be supplied to procedures such as `Tk_SetOptions`, and it supports the TK_OPTION_NULL_OK flag.

**TK_OPTION_BOOLEAN**
The value must be a standard boolean value such as `true` or `no`. The internal form is an integer with value 0 or 1.

**TK_OPTION_BORDER**
The value must be a standard color name such as `red` or `#ff8080`. The internal form is a Tk_3DBorder token like the ones returned by `Tk_Alloc3DBorderFromObj`. This option type requires `tkwin` to be supplied to procedures such as `Tk_SetOptions`, and it supports the TK_OPTION_NULL_OK flag.

**TK_OPTION_COLOR**
The value must be a standard color name such as `red` or `#ff8080`. The internal form is an (XColor *) token like the ones returned by `Tk_AllocColorFromObj`. This option type requires `tkwin` to be supplied to procedures such as `Tk_SetOptions`, and it supports the TK_OPTION_NULL_OK flag.

**TK_OPTION_CURSOR**
The value must be a standard cursor name such as `cross` or `@foo`. The internal form is a Tk_Cursor token like the ones returned by `Tk_AllocCursorFromObj`. This option type requires `tkwin` to be supplied to procedures such as `Tk_SetOptions`, and when the option is set the cursor for the window is changed by calling `XDefineCursor`. This option type also supports the TK_OPTION_NULL_OK flag.

**TK_OPTION_CUSTOM**
This option allows applications to define new option types. The clientData field of the entry points to a structure defining the new option type. See the section CUSTOM OPTION TYPES below for details.

**TK_OPTION_DOUBLE**
The string value must be a floating-point number in the format accepted by `strtol`. The internal form is a C `double` value. This option type supports the TK_OPTION_NULL_OK flag; if a NULL value is set, the internal representation is set to zero.

**TK_OPTION_END**
Marks the end of the template. There must be a Tk_OptionSpec structure with `type TK_OPTION_END` at the end of each template. If the `clientData` field of this structure isn't NULL, then it points to an additional array of Tk_OptionSpec's, which is itself terminated by another `TK_OPTION_END` entry. Templates may be chained arbitrarily deeply. This feature allows common options to be shared by several widget classes.

**TK_OPTION_FONT**
The value must be a standard font name such as `Times 16`. The internal form is a Tk_Font handle like the ones returned by `Tk_AllocFontFromObj`. This option type requires `tkwin` to be supplied to procedures such as `Tk_SetOptions`, and it supports the TK_OPTION_NULL_OK flag.

**TK_OPTION_INT**
The string value must be an integer in the format accepted by `strtol` (e.g. 0 and 0x prefixes may be used to specify octal or hexadecimal numbers, respectively). The internal form is a C `int` value.
**TK_OPTION_JUSTIFY**

The value must be a standard justification value such as `left`. The internal form is a Tk_Justify like the values returned by `Tk_GetJustifyFromObj`.

**TK_OPTION_PIXELS**

The value must specify a screen distance such as `2i` or `6.4`. The internal form is an integer value giving a distance in pixels, like the values returned by `Tk_GetPixelsFromObj`. Note: if the `objOffset` field isn't used then information about the original value of this option will be lost. See **OBJOFFSET VS. INTERNALOFFSET** below for details. This option type supports the `TK_OPTION_NULL_OK` flag; if a NULL value is set, the internal representation is set to zero.

**TK_OPTION_RELIEF**

The value must be standard relief such as `raised`. The internal form is an integer relief value such as `TK_RELIEF_RAISED`. This option type supports the `TK_OPTION_NULL_OK` flag; if the empty string is specified as the value for the option, the integer relief value is set to `TK_RELIEF_NULL`.

**TK_OPTION_STRING**

The value may be any string. The internal form is a (char *) pointer that points to a dynamically allocated copy of the value. This option type supports the `TK_OPTION_NULL_OK` flag.

**TK_OPTION_STRING_TABLE**

For this type, `clientData` is a pointer to an array of strings suitable for passing to `Tcl_GetIndexFromObj`. The value must be one of the strings in the table, or a unique abbreviation of one of the strings. The internal form is an integer giving the index into the table of the matching string, like the return value from `Tcl_GetStringFromObj`.

**TK_OPTION_SYNONYM**

This type is used to provide alternative names for an option (for example, `-bg` is often used as a synonym for `-background`). The `clientData` field is a (char *) pointer that gives the name of another option in the same table. Whenever the synonym option is used, the information from the other option will be used instead.

**TK_OPTION_WINDOW**

The value must be a window path name. The internal form is a Tk_Window token for the window. This option type requires `tkwin` to be supplied to procedures such as `Tk_SetOptions` (in order to identify the application), and it supports the `TK_OPTION_NULL_OK` flag.

### STORAGE MANAGEMENT ISSUES

If a field of a widget record has its offset stored in the `objOffset` or `internalOffset` field of a Tk_OptionSpec structure then the procedures described here will handle all of the storage allocation and resource management issues associated with the field. When the value of an option is changed, `Tk_SetOptions` (or `Tk_FreeSavedOptions`) will automatically free any resources associated with the old value, such as Tk_Fonts for `TK_OPTION_FONT` options or dynamically allocated memory for `TK_OPTION_STRING` options. For an option stored as an object using the `objOffset` field of a Tk_OptionSpec, the widget record shares the object pointed to by the `objv` value from the call to `Tk_SetOptions`. The reference count for this object is incremented when a pointer to it is stored in the widget record and decremented when the option is modified. When the widget is deleted `Tk_FreeConfigOptions` should be invoked; it will free the resources associated with all options and decrement reference counts for any objects.

However, the widget code is responsible for storing NULL or `None` in all pointer and token fields before
invoking **Tk_InitOptions**. This is needed to allow proper cleanup in the rare case where an error occurs in **Tk_InitOptions**.

**OBJOFFSET VS. INTERNALOFFSET**

In most cases it is simplest to use the `internalOffset` field of a Tk_OptionSpec structure and not the `objOffset` field. This makes the internal form of the value immediately available to the widget code so the value doesn't have to be extracted from an object each time it is used. However, there are two cases where the `objOffset` field is useful. The first case is for TK_OPTION_PIXELS options. In this case, the internal form is an integer pixel value that is valid only for a particular screen. If the value of the option is retrieved, it will be returned as a simple number. For example, after the command `.b configure –borderwidth 2m`, the command `.b configure –borderwidth` might return 7, which is the integer pixel value corresponding to `2m`. Unfortunately, this loses the original screen-independent value. Thus for TK_OPTION_PIXELS options it is better to use the `objOffset` field. In this case the original value of the option is retained in the object and can be returned when the option is retrieved. In most cases it is convenient to use the `internalOffset` field as well, so that the integer value is immediately available for use in the widget code (alternatively, **Tk_GetPixelsFromObj** can be used to extract the integer value from the object whenever it is needed). Note: the problem of losing information on retrievals exists only for TK_OPTION_PIXELS options.

The second reason to use the `objOffset` field is in order to implement new types of options not supported by these procedures. To implement a new type of option, you can use TK_OPTION_STRING as the type in the Tk_OptionSpec structure and set the `objOffset` field but not the `internalOffset` field. Then, after calling **Tk_SetOptions**, convert the object to internal form yourself.

**CUSTOM OPTION TYPES**

Applications can extend the built-in configuration types with additional configuration types by writing procedures to parse, print, free, and restore saved copies of the type and creating a structure pointing to those procedures:

```c
typedef struct Tk_ObjCustomOption {
  char *name;
  Tk_CustomOptionSetProc *setProc;
  Tk_CustomOptionGetProc *getProc;
  Tk_CustomOptionRestoreProc *restoreProc;
  Tk_CustomOptionFreeProc *freeProc;
  ClientData clientData;
} Tk_ObjCustomOption;

typedef int Tk_CustomOptionSetProc(
  ClientData clientData,
  Tcl_Interp *interp,
  Tk_Window tkwin,
  Tcl_Obj **valuePtr,
  char *recordPtr,
  int internalOffset,
  char *saveInternalPtr,
  int flag);
```

The code above is a partial representation of the structure and the function to set a custom option. The full documentation and implementation would need to define all the necessary fields and procedures for complete functionality.
typedef Tcl_Obj *Tk_CustomOptionGetProc(
    ClientData, Tk_Window
    recordPtr, internalOffset);

typedef void Tk_CustomOptionRestoreProc(
    ClientData, Tk_Window
    internalPtr, saveInternalPtr);

typedef void Tk_CustomOptionFreeProc(
    ClientData, Tk_Window
    internalPtr);

The Tk_ObjCustomOption structure contains six fields: a name for the custom option type; pointers to the
four procedures; and a clientData value to be passed to those procedures when they are invoked. The
clientData value typically points to a structure containing information that is needed by the procedures when
they are parsing and printing options. RestoreProc and freeProc may be NULL, indicating that no function
should be called for those operations.

The setProc procedure is invoked by Tk_SetOptions to convert a Tcl_Obj into an internal representation and
store the resulting value in the widget record. The arguments are:

cclientData
    A copy of the clientData field in the Tk_ObjCustomOption structure.
ainterp
    A pointer to a Tcl interpreter, used for error reporting.
intkwin
    A copy of the tkwin argument to Tk_SetOptions
ivaluePtr
    A pointer to a reference to a Tcl_Obj describing the new value for the option; it could have been
    specified explicitly in the call to Tk_SetOptions or it could come from the option database or a
default. If the objOffset for the option is non−negative (the option value is stored as a (Tcl_Obj *) in
    the widget record), the Tcl_Obj pointer referenced by valuePtr is the pointer that will be stored at the
    objOffset for the option. SetProc may modify the value if necessary; for example, setProc may
    change the value to NULL to support the TK_OPTION_NULL_OK flag.
irecordPtr
    A pointer to the start of the widget record to modify.
ininternalOffset
    Offset in bytes from the start of the widget record to the location where the internal representation of
    the option value is to be placed.
saveInternalPtr
    A pointer to storage allocated in a Tk_SavedOptions structure for the internal representation of the
    original option value. Before setting the option to its new value, setProc should set the value
referenced by saveInternalPtr to the original value of the option in order to support
Tk_RestoreSavedOptions.

flags

A copy of the flags field in the Tk_OptionSpec structure for the option

SetProc returns a standard Tcl result: TCL_OK to indicate successful processing, or TCL_ERROR to indicate
a failure of any kind. An error message may be left in the Tcl interpreter given by interp in the case of an
error.

The getProc procedure is invoked by Tk_GetOptionValue and Tk_GetOptionInfo to retrieve a Tcl_Obj
representation of the internal representation of an option. The clientData argument is a copy of the clientData
field in the Tk_ObjCustomOption structure. Tkwin is a copy of the tkwin argument to Tk_GetOptionValue or
Tk_GetOptionInfo. RecordPtr is a pointer to the beginning of the widget record to query. InternalOffset is
the offset in bytes from the beginning of the widget record to the location where the internal representation
of the option value is stored. GetProc must return a pointer to a Tcl_Obj representing the value of the option.

The restoreProc procedure is invoked by Tk_RestoreSavedOptions to restore a previously saved internal
representation of a custom option value. The clientData argument is a copy of the clientData field in the
Tk_ObjCustomOption structure. Tkwin is a copy of the tkwin argument to Tk_GetOptionValue or
Tk_GetOptionInfo. InternalPtr is a pointer to the location where internal representation of the option value
is stored. SaveInternalPtr is a pointer to the saved value. RestoreProc must copy the value from
saveInternalPtr to internalPtr to restore the value. RestoreProc need not free any memory associated with
either internalPtr or saveInternalPtr; freeProc will be invoked to free that memory if necessary. RestoreProc
has no return value.

The freeProc procedure is invoked by Tk_SetOptions and Tk_FreeSavedOptions to free any storage
allocated for the internal representation of a custom option. The clientData argument is a copy of the clientData
field in the Tk_ObjCustomOption structure. Tkwin is a copy of the tkwin argument to
Tk_GetOptionValue or Tk_GetOptionInfo. InternalPtr is a pointer to the location where the internal representation
of the option value is stored. The freeProc must free any storage associated with the option.
FreeProc has no return value.

KEYWORDS

anchor, bitmap, boolean, border, color, configuration option, cursor, double, font, integer, justify, pixels,
relief, screen distance, synonym

Copyright © 1998 Sun Microsystems, Inc.
Copyright © 1995-1997 Roger E. Critchlow Jr.
SetVisual

NAME

Tk_SetWindowVisual − change visual characteristics of window

SYNOPSIS

#include <tk.h>

int Tk_SetWindowVisual(tkwin, visual, depth, colormap)

ARGUMENTS

Tk_Window tkwin (in)
   Token for window.
Visual *visual (in)
   New visual type to use for tkwin.
int depth (in)
   Number of bits per pixel desired for tkwin.
Colormap colormap (in)
   New colormap for tkwin, which must be compatible with visual and depth.

DESCRIPTION

When Tk creates a new window it assigns it the default visual characteristics (visual, depth, and colormap) for its screen. Tk_SetWindowVisual may be called to change them. Tk_SetWindowVisual must be called before the window has actually been created in X (e.g. before Tk_MapWindow or Tk_MakeWindowExist has been invoked for the window). The safest thing is to call Tk_SetWindowVisual immediately after calling Tk_CreateWindow. If tkwin has already been created before Tk_SetWindowVisual is called then it returns 0 and doesn't make any changes; otherwise it returns 1 to signify that the operation completed successfully.

Note: Tk_SetWindowVisual should not be called if you just want to change a window's colormap without changing its visual or depth; call Tk_SetWindowColormap instead.

KEYWORDS

colormap, depth, visual

Copyright ' 1992 The Regents of the University of California.
Copyright ' 1994-1996 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
StrictMotif

NAME

Tk_StrictMotif − Return value of tk_strictMotif variable

SYNOPSIS

#include <tk.h>

int Tk_StrictMotif(tkwin)

ARGUMENTS

Tk_Window tkwin (in)
    Token for window.

DESCRIPTION

This procedure returns the current value of the tk_strictMotif variable in the interpreter associated with tkwin's application. The value is returned as an integer that is either 0 or 1. 1 means that strict Motif compliance has been requested, so anything that is not part of the Motif specification should be avoided. 0 means that ``Motif−like'' is good enough, and extra features are welcome.

This procedure uses a link to the Tcl variable to provide much faster access to the variable's value than could be had by calling Tcl_GetVar.

KEYWORDS

Motif compliance, tk_strictMotif variable

Copyright ' 1995−1996 Sun Microsystems, Inc.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
WindowId

NAME

Tk_WindowId, Tk_Parent, Tk_Display, Tk_DisplayName, Tk_ScreenNumber, Tk_Screen, Tk_X, Tk_Y, Tk_Width, Tk_Height, Tk_Changes, Tk_Attributes, Tk_IsContainer, Tk_IsEmbedded, Tk_IsMapped, Tk_IsTopLevel, Tk_ReqWidth, Tk_ReqHeight, Tk_MinReqWidth, Tk_MinReqHeight, Tk_InternalBorderLeft, Tk_InternalBorderRight, Tk_InternalBorderTop, Tk_InternalBorderBottom, Tk_Visual, Tk_Depth, Tk_Colormap – retrieve information from Tk's local data structure

SYNOPSIS

#include <tk.h>

Window

Tk_WindowId(tkwin)
Tk_Window

Tk_Parent(tkwin)
Display *

Tk_Display(tkwin)
CONST char *

Tk_DisplayName(tkwin)
int

Tk_ScreenNumber(tkwin)
Screen *

Tk_Screen(tkwin)
int

Tk_X(tkwin)
int

Tk_Y(tkwin)
int

Tk_Width(tkwin)
int

Tk_Height(tkwin)
XWindowChanges *

Tk_Changes(tkwin)
XSetWindowAttributes *

Tk_Attributes(tkwin)
int

Tk_IsContainer(tkwin)
int

Tk_IsEmbedded(tkwin)
int

Tk_IsMapped(tkwin)
int
Tk_IsTopLevel(tkwin)
int
Tk_ReqWidth(tkwin)
int
Tk_ReqHeight(tkwin)
int
Tk_MinReqWidth(tkwin)
int
Tk_MinReqHeight(tkwin)
int
Tk_InternalBorderLeft(tkwin)
int
Tk_InternalBorderRight(tkwin)
int
Tk_InternalBorderTop(tkwin)
int
Tk_InternalBorderBottom(tkwin)
Visual *
Tk_Visual(tkwin)
int
Tk_Depth(tkwin)
Colormap
Tk_Colormap(tkwin)

ARGUMENTS

Tk_Window tkwin (in)
   Token for window.

DESCRIPTION

Tk_WindowId and the other names listed above are all macros that return fields from Tk's local data structure for tkwin. None of these macros requires any interaction with the server; it is safe to assume that all are fast.

Tk_WindowId returns the X identifier for tkwin, or NULL if no X window has been created for tkwin yet.

Tk_Parent returns Tk's token for the logical parent of tkwin. The parent is the token that was specified when tkwin was created, or NULL for main windows.

Tk_Display returns a pointer to the Xlib display structure corresponding to tkwin. Tk_DisplayName returns an ASCII string identifying tkwin's display. Tk_ScreenNumber returns the index of tkwin's screen among all the screens of tkwin's display. Tk_Screen returns a pointer to the Xlib structure corresponding to tkwin's screen.
Tk_X, Tk_Y, Tk_Width, and Tk_Height return information about tkwin's location within its parent and its size. The location information refers to the upper-left pixel in the window, or its border if there is one. The width and height information refers to the interior size of the window, not including any border. Tk_Changes returns a pointer to a structure containing all of the above information plus a few other fields. Tk_Attributes returns a pointer to an XSetWindowAttributes structure describing all of the attributes of the tkwin's window, such as background pixmap, event mask, and so on (Tk keeps track of all this information as it is changed by the application). Note: it is essential that applications use Tk procedures like Tk_ResizeWindow instead of X procedures like XResizeWindow, so that Tk can keep its data structures up-to-date.

Tk_IsContainer returns a non-zero value if tkwin is a container, and that some other application may be embedding itself inside tkwin.

Tk_IsEmbedded returns a non-zero value if tkwin is is not a free-standin window, but rather is embedded in some other application.

Tk_IsMapped returns a non-zero value if tkwin is mapped and zero if tkwin isn't mapped.

Tk_IsTopLevel returns a non-zero value if tkwin is a top-level window (its X parent is the root window of the screen) and zero if tkwin isn't a top-level window.

Tk_ReqWidth and Tk_ReqHeight return information about the window's requested size. These values correspond to the last call to Tk_GeometryRequest for tkwin.

Tk_MinReqWidth and Tk_MinReqHeight return information about the window's minimum requested size. These values correspond to the last call to Tk_SetMinimumRequestSize for tkwin.

Tk_InternalBorderLeft, Tk_InternalBorderRight, Tk_InternalBorderTop and Tk_InternalBorderBottom return the width of one side of the internal border that has been requested for tkwin, or 0 if no internal border was requested. The return value is simply the last value passed to Tk_SetInternalBorder or Tk_SetInternalBorderEx for tkwin.

Tk_Visual, Tk_Depth, and Tk_Colormap return information about the visual characteristics of a window. Tk_Visual returns the visual type for the window, Tk_Depth returns the number of bits per pixel, and Tk_Colormap returns the current colormap for the window. The visual characteristics are normally set from the defaults for the window's screen, but they may be overridden by calling Tk_SetWindowVisual.

KEYWORDS

attributes, colormap, depth, display, height, geometry manager, identifier, mapped, requested size, screen, top-level, visual, width, window, x, y

Copyright ' 1990-1993 The Regents of the University of California.
Copyright ' 1994-1997 Sun Microsystems, Inc.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
Tcl/Tk Keywords

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Copyright ' 1989-1994 The Regents of the University of California.
Copyright ' 1990-1994 The Regents of the University of California
Copyright ' 1992-1999 Karl Lehenbauer and Mark Diekhans
Copyright ' 1992-1999 Karl Lehenbauer and Mark Diekhans.
Copyright ' 1993-1997 Bell Labs Innovations for Lucent Technologies
Copyright ' 1994 The Australian National University
Copyright ' 1994-2000 Sun Microsystems, Inc.
Copyright ' 1997-2000 Ajuba Solutions.
Copyright ' 1997-2000 Scriptics Corporation.
Copyright ' 1998 Mark Harrison.
Copyright ' 1998-1999 Scriptics Corporation
Copyright ' 1998-2000 Ajuba Solutions
Copyright ' 1999 Scriptics Corporation
Copyright ' 2000 Jeffrey Hobbs.
Copyright ' 2001 ActiveState Tool Corp.
Copyright ' 2001 Kevin B. Kenny.
Copyright ' 2001 Vincent Darley
Copyright ' 2001-2002 ActiveState Corporation
Copyright ' 2001-2002 Donal K. Fellows
Copyright ' 2001-2002 Kevin B. Kenny. All rights reserved.
Copyright ' 2001-2004 ActiveState Corporation.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
Tcl/Tk Keywords – A

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

abort
break, Panic
above
Restack
absolute file name
filename
access
Access
access mode
open
access point
OpenFileChnl
access position
seek, tell
add
incr, AddOption
alias
interp, safe, loadTk, CrtSlave
alloc
Alloc
allocation
Alloc
alpha
GetVersion
anchor
ConfigWidg, SetOptions
anchor position
GetAnchor
append
append, lappend, open, clipboard, DString, ListObj, SetResult, StringObj, Clipboard
application
dde, destroy, send, AppInit, Init, MainLoop, MainWin, SetAppName, Tk_Init
application name
tk
application−specific initialization
SourceRCFile, Tcl_Main, Tk_Main
argument	clsh, proc, AppInit
arguments
ParseArgv
arithmetic
   expr, tclvars
array
   array, SetVar
aspect ratio
   wm
association
   AssocData
asynchronous event
   Async
asynchronous I/O
   fileevent
atom
   winfo, GetUid, InternAtom
attributes
   file, ConfigWind, WindowId
auto−exec
   library
auto−load
   library, packagens, pkgMkIndex
auto−loading
   safe, loadTk
auto_mkindex
   safe, loadTk
Tcl/Tk Keywords – B

**background**
- **BackgdErr**, **DetachPids**, **3DBorder**

**background error**
- **bgerror**, **tkerror**

**backslash**
- **Backslash**, **SplitList**, **Utf**

**backslash substitution**
- **subst**, **ParseCmd**, **SubstObj**

**beep**
- **bell**

**bell**
- **bell**

**below**
- **Restack**

**beta**
- **GetVersion**

**bevel**
- **GetJoinStl**

**bgerror**
- **BackgdErr**

**binary**
- **binary**, **fconfigure**, **FindExec**

**binary code**
- **load**

**bind**
- **socket**, **keysyms**, **CrtCommand**, **CrtObjCmd**, **CrtCmHdlr**, **CrtGenHdlr**, **EventHndlr**

**binding**
- **bind**, **bindtags**, **event**, **keysyms**, **BindTable**

**bisque**
- **palette**

**bitmap**
- **bitmap**, **dialog**, **CanvPsY**, **ConfigWidg**, **GetBitmap**, **SetOptions**

**blocking**
- **close**, **fblocked**, **fconfigure**, **fcopy**, **fileevent**, **flush**, **gets**, **read**, **CrtChannel**, **CrtChnlHdlr**, **OpenFileChnl**

**boolean**
- **expr**, **if**, **BoolObj**, **ExprLong**, **ExprLongObj**, **GetInt**, **LinkVar**, **ConfigWidg**, **SetOptions**

**boolean object**
- **BoolObj**

**boolean type**
BoolObj
boolean value
while
border
3DBorder, ConfigWdg, ConfigWind, SetOptions
braces
ParseCmd
break
break, return, AllowExc
buffer
flush
buffered I/O
OpenFileChnl
buffering
fconfigure
butt
GetCapStyl
button
button
byte array
fconfigure, ByteArrObj
bytecode
tclvars
Tcl/Tk Keywords – C

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

cache
callback
    CallDel, CrtChnlHdlr, CrtCloseHdlr, CrtFileHdlr, CrtTimerHdlr, DoOneEvent, DoWhenIdle, Exit, CrtCmHdlr, CrtErrHdlr, CrtGenHdlr, EventHndlr, HandleEvent, ManageGeom, QWinEvent, SetClassProcs
cancel
    after
canvas
    canvas, CanvPsY, CanvTkwin, CanvTxtInfo, CrtItemType
cap style
    ConfigWidg, GetCapStyl
caret
    SetCaret
carriage return
    fconfigure
case
    ToUpper
case conversion
    string
catch
    catch, return
cell
    grid
center
    GetJustify
centimeters
    GetPixels
channel
    close, eof, fcopy, fileevent, flush, gets, puts, read, socket, tell, ChnlStack, CrtChnlHdlr, GetOpnFl, OpenFileChnl
channel closing
    CrtCloseHdlr
channel driver
    CrtChannel, OpenFileChnl
channel registration
    CrtChannel
channel type
    CrtChannel
checkbutton
checkbutton
child
DetachPids
children

winfo

class
options, winfo, AddOption, GetOption, SetClass, SetClassProcs

classification
UniCharIsAlpha

cleanup
Exit

clear
clipboard, selection, Clipboard, ChrSelect

client

OpenTcp

clientData
TraceCmd, TraceVar

clipboard

clipboard, Clipboard

clock
clock, CrtTimerHdlr, QWinEvent

close
close

color
colors, palette, photo, 3DBorder, CanvPsY, ConfigWidg, ConfigWind, GetColor, SetOptions

color selection dialog
chooseColor

colormap
GetClrmap, GetVisual, SetVisual, WindowId

command
info, rename, trace, AppInit, CrtCommand, CrtInterp, CrtObjCmd, CrtSlave, CrtTrace, ParseCmd, RecEvalObj, RecordEval, SetResult, TraceCmd, WrongNumArgs

command line
ParseArgv

command substitution

subst, SubstObj

command−line arguments
Tcl_Main, Tk_Main

compare
expr, string

compiler
tclvars

complete command

CmdCmplt

compression
ChnlStack
concat
StringObj
concatenate
concat, eval, Concat, StringObj
condition variable
Thread
conditional
if
configuration option
SetOptions
configuration options
ConfigWidg
configure
ConfigWind
connection
socket
console
console
containing
CoordToWin
context
uplevel, upvar
continue
continue, return, AllowExc
conversion
GetInt, PrintDbl, GetDash
conversion specifier
format, scan
convert
Encoding, SplitList, GetPixels
coordinates
CoordToWin, GetRootCrd
copy files
file
create
open, CrtCommand, CrtInterp, CrtObjCmd, CrtTrace, CrtWindow
cstype
string
current directory
filename
cursor
cursors, ConfigWidg, GetCursor, SetCaret, SetOptions
custom
ConfigWidg
Tcl/Tk Keywords – D

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

dash  GetDash

data  AssocData

database  option

date  clock, GetTime

dde  dde, send

debug  memory, DumpActiveMemory, TCL_MEM_DEBUG

defer  DoWhenIdle

defered creation  CrtWindow

define  event

deiconify  wm

delay  after, RestrictEv

delete  rename, CallDel, CrtCommand, CrtInterp, CrtObjCmd, CrtTrace

delete files  file

delete image  DeleteImg

deletion procedure  AssocData

depressed  3DBorder

depth  SetVisual, WindowId

destroy  destroy, CrtWindow

detach  DetachPids

dialog  chooseDirectory, dialog
directory
  file, chooseDirectory
display
  CrtWindow, InternAtom, WindowId
domain name
  socket
double
  DoubleObj, ExprLong, ExprLongObj, GetInt, ConfigWidg, SetOptions
double object
  DoubleObj
double type
  DoubleObj
double-precision
  PrintDbl
dynamic loading
  Exit
dynamic string
  DString

Copyright ' 1989−1994 The Regents of the University of California.
Copyright ' 1990−1994 The Regents of the University of California
Copyright ' 1992−1999 Karl Lehenbauer and Mark Diekhans
Copyright ' 1992−1999 Karl Lehenbauer and Mark Diekhans.
Copyright ' 1993−1997 Bell Labs Innovations for Lucent Technologies
Copyright ' 1994 The Australian National University
Copyright ' 1994−2000 Sun Microsystems, Inc.
Copyright ' 1997−2000 Ajuba Solutions.
Copyright ' 1997−2000 Scriptics Corporation.
Copyright ' 1998 Mark Harrison.
Copyright ' 1998−1999 Scriptics Corporation
Copyright ' 1998−2000 Ajuba Solutions
Copyright ' 1999 Scriptics Corporation
Copyright ' 2000 Jeffrey Hobbs.
Copyright ' 2001 ActiveState Tool Corp.
Copyright ' 2001 Kevin B. Kenny.
Copyright ' 2001 Vincent Darley
Copyright ' 2001−2002 ActiveState Corporation
Copyright ' 2001−2002 Donal K. Fellows
Copyright ' 2001−2002 Kevin B. Kenny. All rights reserved.
Copyright ' 2001−2004 ActiveState Corporation.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
Tcl/Tk Keywords – E

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

element
join, lappend, lindex, linsert, list, llength, lrange, lreplace, lset, lsort, SetResult, SplitList
element names
array
else
if
encoding
encoding, fconfigure, read, Encoding
end application
Exit
end of file
eof, fcopy, gets, read, OpenFileChnl
end of line
fconfigure, fcopy, gets, read
entry
entry, spinbox
environment
tclvars, Environment
equal
string
ero
SetErno
error
catch, error, return, tclvars, unknown, AddErrInfo, BackgdErr, Panic, CrtErrHdlr
error code
SetErno
error message
WrongNumArgs
errorCode
error
errorInfo
error
evaluate
eval, ExprLong, ExprLongObj
event
history, update, ywait, bind, bindtags, event, DoOneEvent, Notifier, RecEvalObj, RecordEval,
BindTable, CrtCmHdlr, CrtErrHdlr, CrtGenHdlr, EventHndlr, HandleEvent, MainLoop, RestrictEv
event handler
fileevent
event queue
Notifier
event sources
Notifier
events
  focus, CrtChnlHdlr, QWinEvent
exception
  AllowExc
executable file
  FindExec
execute
  exec, Eval, RecEvalObj, RecordEval
exist
  glob
exit
  exit, Exit
exported
namespace
exposed commands
  CrtSlave
expression
  expr, CrtMathFnc, ExprLong, ExprLongObj, ParseCmd

Copyright ' 1989−1994 The Regents of the University of California.
Copyright ' 1990−1994 The Regents of the University of California
Copyright ' 1992−1999 Karl Lehenbauer and Mark Diekhans
Copyright ' 1992−1999 Karl Lehenbauer and Mark Diekhans.
Copyright ' 1993−1997 Bell Labs Innovations for Lucent Technologies
Copyright ' 1994 The Australian National University
Copyright ' 1994−2000 Sun Microsystems, Inc.
Copyright ' 1997−2000 Ajuba Solutions.
Copyright ' 1997−2000 Scriptics Corporation.
Copyright ' 1998 Mark Harrison.
Copyright ' 1998−1999 Scriptics Corporation
Copyright ' 1998−2000 Ajuba Solutions
Copyright ' 1999 Scriptics Corporation
Copyright ' 2000 Jeffrey Hobbs.
Copyright ' 2001 ActiveState Tool Corp.
Copyright ' 2001 Kevin B. Kenny.
Copyright ' 2001 Vincent Darley
Copyright ' 2001−2002 ActiveState Corporation
Copyright ' 2001−2002 Donal K. Fellows
Copyright ' 2001−2002 Kevin B. Kenny. All rights reserved.
Copyright ' 2001−2004 ActiveState Corporation.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
false

if

fatal

Panic

file

file, glob, open, pid, seek, source, CrtFileHdlr, Eval, SplitPath

file events

Notifier

file handle

GetOpenFl

file name

Translate

file selection dialog

getOpenFile

filename

SplitPath

fill

GetJustify

filter

fconfigure, RestrictEv

floating-point

GetInt, PrintDbl

flush

flush, update, OpenFileChnl

flushing

fconfigure

focus

focus, focusNext, CanvTkwin, CanvTxtInfo, CrtItemType, DrawFocHlt

focus model

wm

font

font, CanvPsY, ConfigWidg, FontId, GetFont, MeasureChar, SetOptions, TextLayout

for

for

foreach

foreach

format

binary, format, clipboard, selection, Clipboard, CrtSelHdlr, GetSelect

frame

upvar, frame
free Alloc, DString, Interp, Preserve fuzzy comparison expr
Tcl/Tk Keywords – G

geometry
  winfo, wm, GeomReq, GetVRoot, ManageGeom

geometry management
  panedwindow

geometry manager
  grid, pack, place, MaintGeom, WindowId

get
  GetSelect

get variable
  SetVar

glob
  glob

global
  global, upvar, variable, Eval

global variables
  SetErrno

grab
  grab, Grab

graphics context
  GetGC

grid
  grid, wm, SetGrid

group
  wm

Copyright ' 1989−1994 The Regents of the University of California.
Copyright ' 1990−1994 The Regents of the University of California
Copyright ' 1992−1999 Karl Lehenbauer and Mark Diekhans
Copyright ' 1992−1999 Karl Lehenbauer and Mark Diekhans.
Copyright ' 1993−1997 Bell Labs Innovations for Lucent Technologies
Copyright ' 1994 The Australian National University
Copyright ' 1994−2000 Sun Microsystems, Inc.
Copyright ' 1997−2000 Ajuba Solutions.
Copyright ' 1997−2000 Scriptics Corporation.
Copyright ' 1998 Mark Harrison.
Copyright ' 1998−1999 Scriptics Corporation
Copyright ' 1998−2000 Ajuba Solutions
Copyright ' 1999 Scriptics Corporation
Copyright ' 2000 Jeffrey Hobbs.
Copyright ' 2001 ActiveState Tool Corp.
Copyright ' 2001 Kevin B. Kenny.
Copyright ' 2001 Vincent Darley
Copyright ' 2001−2002 ActiveState Corporation
Tcl/Tk Keywords – H

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

handle

event

handler
update, selection, Async, CrtChnlHdlr, CrtFileHdlr, CrtTimerHdlr, DoOneEvent, CrtCmHdlr,
CrtErrHdlr, CrtGenHdlr, CrtSelHdlr, EventHndlr, HandleEvent, QWinEvent

hash table
Hash

height
image, place, winfo, ConfigWind, GetVRoot, WindowId

hidden commands
CrtSlave

history
history, RecEvalObj, RecordEval

home directory
Translate

host
socket

hostname
GetHostName

Copyright ' 1989-1994 The Regents of the University of California.
Copyright ' 1990-1994 The Regents of the University of California
Copyright ' 1992-1999 Karl Lehenbauer and Mark Diekhans
Copyright ' 1992-1999 Karl Lehenbauer and Mark Diekhans.
Copyright ' 1993-1997 Bell Labs Innovations for Lucent Technologies
Copyright ' 1994 The Australian National University
Copyright ' 1994-2000 Sun Microsystems, Inc.
Copyright ' 1997-2000 Ajuba Solutions.
Copyright ' 1997-2000 Scriptics Corporation.
Copyright ' 1998 Mark Harrison.
Copyright ' 1998-1999 Scriptics Corporation
Copyright ' 1998-2000 Ajuba Solutions
Copyright ' 1999 Scriptics Corporation
Copyright ' 2000 Jeffrey Hobbs.
Copyright ' 2001 ActiveState Tool Corp.
Copyright ' 2001 Kevin B. Kenny.
Copyright ' 2001 Vincent Darley
Copyright ' 2001-2002 ActiveState Corporation
Copyright ' 2001-2002 Donal K. Fellows
Copyright ' 2001-2002 Kevin B. Kenny. All rights reserved.
Copyright ' 2001-2004 ActiveState Corporation.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
Tcl/Tk Keywords – I

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

i18n
  msgcat
ICCCM
  selection
icon
  wm
iconify
  wm
identifier
  winfo, GetHINSTANCE, GetHWND, WindowId
idle
  update, DoOneEvent, Notifier
idle callback
  after, DoWhenIdle
if
  if
illumination
  3DBorder
image
  bitmap, image, photo, FindPhoto
image file
  CrtPhImgFmt
image manager
  CrtImgType, DeleteImg, NameOfImg
image name
  NameOfImg
image size changes
  ImgChanged
image type
  CrtImgType
images
  GetImage, ImgChanged
inches
  GetPixels
increment
  incr
increments
  wm
index
  lindex, lset, packagens, pkgMkIndex, string, GetIndex, ListObj
information  info, winfo
initialization  AppInit, Init, Tk_Init
initialization procedure  StaticPkg
initialized  Interp
input  OpenFileChnl
insert  linsert, ListObj
insertion cursor  CanvTxtInfo
instance  CrtImgType, GetHINSTANCE
integer  ExprLong, ExprLongObj, GetInt, IntObj, LinkVar, ConfigWidg, SetOptions
integer object  IntObj
integer type  IntObj
intensity  GetColor
interactive  console
internal  namespace
internal representation  BoolObj, DoubleObj, IntObj, ListObj, Object, ObjectType, StringObj
internal window  CrtWindow
internationalization  msgcat, ByteArrObj
interp  SaveResult
interpreter  tclsh, info, console, AllowExc, AppInit, AssocData, CallDel, CrtCommand, CrtInterp, CrtSlave, CrtTrace, Init, Interp, RecEvalObj, RecordEval, SetResult, SetVar
interpreters  winfo
invoke  CrtSlave
item type  CanvTkwin, CrtItemType
iteration
  continue, for, foreach
Tcl/Tk Keywords – J

join
concat, join, SplitPath

join style
ConfigWidg, GetJoinStl

justification
GetJustify

justify
ConfigWidg, SetOptions

Copyright ' 1989-1994 The Regents of the University of California.
Copyright ' 1990-1994 The Regents of the University of California
Copyright ' 1992-1999 Karl Lehenbauer and Mark Diekhans
Copyright ' 1992-1999 Karl Lehenbauer and Mark Diekhans.
Copyright ' 1993-1997 Bell Labs Innovations for Lucent Technologies
Copyright ' 1994 The Australian National University
Copyright ' 1994-2000 Sun Microsystems, Inc.
Copyright ' 1997-2000 Ajuba Solutions.
Copyright ' 1997-2000 Scriptics Corporation.
Copyright ' 1998 Mark Harrison.
Copyright ' 1998-1999 Scriptics Corporation
Copyright ' 1998-2000 Ajuba Solutions
Copyright ' 1999 Scriptics Corporation
Copyright ' 2000 Jeffrey Hobbs.
Copyright ' 2001 ActiveState Tool Corp.
Copyright ' 2001 Kevin B. Kenny.
Copyright ' 2001 Vincent Darley
Copyright ' 2001-2002 ActiveState Corporation
Copyright ' 2001-2002 Donal K. Fellows
Copyright ' 2001-2002 Kevin B. Kenny. All rights reserved.
Copyright ' 2001-2004 ActiveState Corporation.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
Tcl/Tk Keywords – K

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

key
  AssocData, Hash
keyboard
  focus
keyboard events
  grab
keyboard traversal
  focusNext
keysym
  keysyms

Copyright ' 1989−1994 The Regents of the University of California.
Copyright ' 1990−1994 The Regents of the University of California.
Copyright ' 1992−1999 Karl Lehenbauer and Mark Diekhans.
Copyright ' 1992−1999 Karl Lehenbauer and Mark Diekhans.
Copyright ' 1993−1997 Bell Labs Innovations for Lucent Technologies.
Copyright ' 1994 The Australian National University.
Copyright ' 1994−2000 Sun Microsystems, Inc.
Copyright ' 1997−2000 Ajuba Solutions.
Copyright ' 1997−2000 Scriptics Corporation.
Copyright ' 1998 Mark Harrison.
Copyright ' 1998−1999 Scriptics Corporation.
Copyright ' 1998−2000 Ajuba Solutions.
Copyright ' 1999 Scriptics Corporation.
Copyright ' 2000 Jeffrey Hobbs.
Copyright ' 2001 ActiveState Tool Corp.
Copyright ' 2001 Kevin B. Kenny.
Copyright ' 2001 Vincent Darley.
Copyright ' 2001−2002 ActiveState Corporation.
Copyright ' 2001−2002 Donal K. Fellows.
Copyright ' 2001−2002 Kevin B. Kenny. All rights reserved.
Copyright ' 2001−2004 ActiveState Corporation.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
Tcl/Tk Keywords – L

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

l10n  msgcat
label  label
labelframe  labelframe
length  llength, ListObj
level  info, uplevel, upvar
library  library
line  gets
linemode  fconfigure
link  LinkVar
linked variable  UpVar
list  foreach, join, lappend, lindex, linsert, list, llength, lrange, lreplace, lsearch, lset, lsort, split, ListObj, SetResult, SplitList
list object  ListObj
list type  ListObj
listbox  listbox
lists  concat
load  safe, loadTk, Tk_Init
loading  load
localization  msgcat
location  grid, pack, place, GetVRoot
lookup
Hash

loop

break, continue, while

looping

for, foreach

lower

lower
Tcl/Tk Keywords – M

mac
Macintosh
macintosh
Macintosh
main loop
MainLoop
main program
SourceRCFile, Tcl_Main, Tk_Main
main window
MainWin, Tk_Init
major
GetVersion
malloc
Alloc, Interp
managed
ManageGeom
map
MaintGeom, MapWindow
mapped
winfo, WindowId
master
place, CrtSlave, CrtImgType, MaintGeom
master interpreter
interp, safe, loadTk
match
lsearch, re_syntax, regexp, regsub, string, switch, RegExp, StrMatch
mathematical function
CrtMathFnc
memory
memory, Alloc, DumpActiveMemory, TCL_MEM_DEBUG
menu
menu, popup
menubutton
menubutton
merge
SplitList
message
msgcat, message
message box
messageBox
millimeters

ConfigWidg, GetPixels

minor

GetVersion

miter

GetJoinStil

modal

dialog

modal timeout

QWinEvent

Motif compliance

StrictMotif

move files

file

mutex

Thread
Tcl/Tk Keywords – N

**name**
dde, file, options, send, AddOption, GetOption, GetRelief, Name, SetAppName

**namespace**
global, info, rename, uplevel, upvar, variable, CrtCommand, CrtObjCmd

**nesting depth**
SetRecLmt

**network address**
socket

**newline**
fconfigure, puts

**non–blocking**
open

**non–existent command**
unknown

**nonblocking**
close, fblocked, fconfigure, fcopy, fileevent, flush, gets, read, CrtChannel, OpenFileChnl

**notifier**
Macintosh, Notifier
Tcl/Tk Keywords – O

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

object
- AddErrInfo, BoolObj, ByteArrObj, CrtObjCmd, DoubleObj, Eval, ExprLong, ExprLongObj,
- GetIndex, IntObj, ListObj, Object, ObjectType, RecEvalObj, SetResult, SetVar, StringObj, 3DBorder,
- BindTable, GetColor

object creation
- Object

object result
- AddErrInfo

object type
- BoolObj, DoubleObj, IntObj, ListObj, Object, ObjectType, StringObj

obscure
- lower, raise, Restack

open
- open, resource

option
- colors, cursors, option, AddOption, GetOption

option menu
- optionMenu

options
- ParseArgv

order
- lsort

output
- flush, puts, OpenFileChnl

output channels
- console

own
- selection, OwnSelect

Copyright ' 1989–1994 The Regents of the University of California.
Copyright ' 1990–1994 The Regents of the University of California
Copyright ' 1992–1999 Karl Lehenbauer and Mark Diekhans
Copyright ' 1992–1999 Karl Lehenbauer and Mark Diekhans.
Copyright ' 1993–1997 Bell Labs Innovations for Lucent Technologies
Copyright ' 1994 The Australian National University
Copyright ' 1994–2000 Sun Microsystems, Inc.
Copyright ' 1997–2000 Ajuba Solutions.
Copyright ' 1998 Mark Harrison.
Copyright ' 1998–1999 Scriptics Corporation
Copyright ' 1998–2000 Ajuba Solutions
Copyright ' 1999 Scriptics Corporation
Tcl/Tk Keywords – P

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

pack
  grid
package
  package, packagens, pkgMkIndex, PkgRequire, StaticPkg
packer
  pack
palette
  palette
panedwindow
  panedwindow
parcel
  pack
parent
  winfo, MaintGeom
parse
  scan, Backslash, ParseCmd, GetScroll
partial command
  CmdCmplt
patchlevel
  GetVersion
path
  SplitPath, CanvPsY
path name
  winfo, Name
pattern
  glob, lsearch, regsub, string, RegExp, StrMatch
permissions
  open, GetOpnFl
photo
  photo, FindPhoto
photo image
  CrtPhImgFmt
pipeline
  exec, open, pid, GetOpnFl
pixel
  ConfigWind
pixel value
  GetColor
pixels
  ConfigWidg, GetPixels, SetOptions
pixmap  ConfigWind, GetBitmap, GetPixmap
place  place
platform  fconfigure
platform−specific  chooseDirectory
pointer events  grab
points  GetPixels
polygon  3DBorder
popup  popup
portability  filename
position  wm, MaintGeom, MoveToplevel
POSIX  tclvars
Postscript  CanvPsY
precision  tclvars
present  PkgRequire
priority  option
procedure  global, info, proc, return, upvar, variable
process  exit, open, DetachPids
process identifier  pid
projecting  GetCapStyl
prompt  tclsh
propagation  grid, pack
provide  PkgRequire
pwd
GetCwd
Tcl/Tk Keywords – R

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

radiobutton
raise
raised
range
rc file
read
read–only
readable
real
realloc
record
recursion
redirection
redisplay
reference count
reference counting
register
registry
regular expression
relative file name
read
rc file
range
radiobutton
raise
raised
3DBorder
lrange
SourceRCFile
fcopy, gets, read, set, trace, GetOpnFl, OpenFileChnl
LinkVar
fileevent
LinkVar
Alloc
history, RecEvalObj, RecordEval
SetRecLmt
exec
CanvTkwin, GetImage, ImgChanged
Preserve
Object
SetAppName
regexp
regexp
RegExp
lssearch, re syntax, regexp, regsub, switch
release  GetVersion
relief  ConfigWidg, GetRelief, SetOptions
remote execution  dde, send
remove  unset
rename  rename, trace
rename files  file
replace  lreplace, lset, ListObj
reporting  bgerror, tkerror
request  GeomReq, ManageGeom
requested size  WindowId
require  PkgRequire
resource  resource, Macintosh
resource identifier  FreeXId, GetPixmap
restriction  RestrictEv
result  DString, Eval, Interp, SaveResult, SetResult
retrieve  option, GetOption
return  return
return value  SetResult
ring  bell
root window  CoordToWin, GetRootCrd
round  GetCapStyl, GetJoinStl
rubber sheet  place

Copyright © 1989-1994 The Regents of the University of California.
Tcl/Tk Keywords − S

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

safe  Tk_Init
safe interpreter  safe, loadTk
safe interpreter  interp, load
scalar  SetVar
scale  scale
scan  binary, scan
screen  wininfo, CrtWindow, GetVisual, WindowId
screen distance  SetOptions
screen units  GetPixels
script  eval, fileevent, source, time, Eval, BindTable
script file  tclsh
scrollbar  scrollbar, GetScroll
scrolling command  GetScroll
search  array, lsearch, Hash
security  send
security policy  http
seek  seek, OpenFileChnl
seeking  tell
selection  chooseDirectory, clipboard, selection, CanvTkwin, CanvTxtInfo, CtrSelect, CrtItemTyp, CrtSelHdr
selection anchor  CanvTxtInfo
selection owner
OwnSelect
selection retrieval
GetSelect
send
send, tk
send command
SetAppName
separator
join
serial
open
server
OpenTcp
service mode
Notifier
set
lset, SetVar
shadow
3DBorder
shared library
load
shell
tclsh, wish
signal
Async
signal numbers
Signal
signals
Signal
size
grid, pack, wm
slave
place, CrtSlave, MaintGeom
slave interpreter
interp, safe, loadTk
sleep
after, Sleep
slider
scale
socket
http, socket
sort
lsort
source
safe, loadTk

spinbox

split

sprintf

stack

AddErrInfo

stack frame

uplevel

stacking order

lower, raise, Restack

standard channel

GetStdChan

standard channels

StdChannels

standard error

GetStdChan

standard input

GetStdChan

standard option

options

standard output

GetStdChan

stat

file, Access

stat access filesystem vfs

FileSystem

state

SaveResult

static linking

StaticPkg

stipple

CanvPsY

storage

Preserve

string

format, lsearch, re_syntax, regexp, split, string, ExprLong, ExprLongObj, LinkVar, PrintDbl, RegExp, StrMatch, GetJustify, GetRelief

string object

StringObj

string representation

BoolObj, DoubleObj, IntObj, ListObj, Object, ObjectType, StringObj

string type
StringObj
strings
   Concat, SplitList
stubs
   InitStubs, TkInitStubs
subexpression
   RegExp
sublist
   lrange
subprocess
   exec, tclvars
substitute
   regsub
substitution
   format
switch
   switch, options
synonym
   ConfigWidg, SetOptions

Copyright ' 1989−1994 The Regents of the University of California.
Copyright ' 1990−1994 The Regents of the University of California
Copyright ' 1992−1999 Karl Lehenbauer and Mark Diekhans
Copyright ' 1992−1999 Karl Lehenbauer and Mark Diekhans.
Copyright ' 1993−1997 Bell Labs Innovations for Lucent Technologies
Copyright ' 1994 The Australian National University
Copyright ' 1994−2000 Sun Microsystems, Inc.
Copyright ' 1997−2000 Ajuba Solutions.
Copyright ' 1997−2000 Scriptics Corporation.
Copyright ' 1998 Mark Harrison.
Copyright ' 1998−1999 Scriptics Corporation
Copyright ' 1998−2000 Ajuba Solutions
Copyright ' 1999 Scriptics Corporation
Copyright ' 2000 Jeffrey Hobbs.
Copyright ' 2001 ActiveState Tool Corp.
Copyright ' 2001 Kevin B. Kenny.
Copyright ' 2001 Vincent Darley
Copyright ' 2001−2002 ActiveState Corporation
Copyright ' 2001−2002 Donal K. Fellows
Copyright ' 2001−2002 Kevin B. Kenny. All rights reserved.
Copyright ' 2001−2004 ActiveState Corporation.
Copyright ' 1995−1997 Roger E. Critchlow Jr.
Tcl/Tk Keywords – T

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

table lookup
  GetIndex

  tag
  bindtags

  target
  selection, CrtSelHdr

TCL_MEM_DEBUG
  Alloc

 Tcl_RegExpIndices
  RegExp

 Tcl_RegExpInfo
  RegExp

tcp
  socket

 TCP
  OpenTcp

test
  tcltest, while

test harness
  tcltest

test suite
  tcltest

text
  msgcat, text, tkvars, CanvTxtInfo

thread
  Exit, Thread

thread local storage
  Thread

threads
  Notifier

three-dimensional effect
  3DBorder

tilde
  Translate

time
  after, clock, time, GetTime, Sleep

timer
  CrtTimerHdr, DoOneEvent, Notifier

title
  wm
tk_strictMotif variable

StrictMotif
tkvars

text
token

ParseCmd, Name
tolower

ToUpper
toolkit

ToUpper
toplevel

ToUpper
toplevel
totitle

ToUpper
toupper

ToUpper
trace

trace, AddErrInfo, CrtTrace, TraceCmd, TraceVar
traces

LinkVar
translate

Translate
translation

fconfigure, fcopy, msgcat, read
traversal highlight

DrawFocHlt
trough

scale
true

if
type

clipboard, selection, SplitPath, Clipboard
type conversion

Object, ObjectType
type manager

CanvTkwin, CrtItemType
types of images

image

Copyright ' 1989−1994 The Regents of the University of California.
Copyright ' 1990−1994 The Regents of the University of California
Copyright ' 1992−1999 Karl Lehenbauer and Mark Diekhans
uid
ConfigWidg
unicode
ByteArrObj, StringObj, ToUpper, UniCharIsAlpha, Utf
unique identifier
GetUid, SetClass
units
wm
unknown
library
unloading
Exit
unmanaged
ManageGeom
unmap
MaintGeom, MapWindow
unset
trace, SetVar
update
update
upvar
UpVar
user
Translate
utf
ByteArrObj, Encoding, ToUpper, Utf
Tcl/Tk Keywords – V

value
    incr, Hash

variable
    append, global, incr, info, lappend, namespace, set, trace, unset, upvar, variable, vwait, tkwait,
    AddErrInfo, Environment, LinkVar, SetVar, TraceVar, UpVar

variable substitution
    subst, ParseCmd, SubstObj

variables
    tclvars, uplevel, tkvars

version
    package, packagens, pkgMkIndex, tkvars, GetVersion, PkgRequire

virtual event
    event

virtual root
    winfo, GetVRoot

visibility
    tkwait

visual
    GetVisual, SetVisual, WindowId

volume–relative file name
    filename

Copyright ' 1989−1994 The Regents of the University of California.
Copyright ' 1990−1994 The Regents of the University of California
Copyright ' 1992−1999 Karl Lehenbauer and Mark Diekhans
Copyright ' 1992−1999 Karl Lehenbauer and Mark Diekhans.
Copyright ' 1993−1997 Bell Labs Innovations for Lucent Technologies
Copyright ' 1994 The Australian National University
Copyright ' 1994−2000 Sun Microsystems, Inc.
Copyright ' 1997−2000 Ajuba Solutions.
Copyright ' 1997−2000 Scriptics Corporation.
Copyright ' 1998 Mark Harrison.
Copyright ' 1998−1999 Scriptics Corporation
Copyright ' 1998−2000 Ajuba Solutions
Copyright ' 1999 Scriptics Corportation
Copyright ' 2000 Jeffrey Hobbs.
Copyright ' 2001 ActiveState Tool Corp.
Copyright ' 2001 Kevin B. Kenny.
Copyright ' 2001 Vincent Darley
Copyright ' 2001−2002 ActiveState Corporation
Copyright ' 2001−2002 Donal K. Fellows
Copyright ' 2001−2002 Kevin B. Kenny. All rights reserved.
Copyright ' 2001−2004 ActiveState Corporation.
Tcl/Tk Keywords – W

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

wait
   vwait, tkwait, DetachPids, Sleep
while
   while
whitespace
   library
widget
   button, canvas, checkbutton, entry, frame, label, labelframe, listbox, menu, menubutton, message, panedwindow, radiobutton, scale, scrollbar, spinbox, text, toplevel
width
   image, place, winfo, ConfigWind, GetVRoot, WindowId
window
   console, destroy, grab, tkwait, winfo, ConfigWind, CrtWindow, GetHWND, Grab, HandleEvent, MapWindow, Name, SetClass, SetGrid, WindowId
window manager
   focus, wm, GetVRoot, MoveToplevel, SetClass, SetGrid
Windows window id
   HWNDToWindow
word
   library, string
working directory
   cd, pwd
writable.
   fileevent
write
   puts, set, trace, GetOpnFl, OpenFileChnl
wrong number of arguments
   WrongNumArgs

Copyright ' 1989−1994 The Regents of the University of California.
Copyright ' 1990−1994 The Regents of the University of California
Copyright ' 1992−1999 Karl Lehenbauer and Mark Diekhans
Copyright ' 1992−1999 Karl Lehenbauer and Mark Diekhans.
Copyright ' 1993−1997 Bell Labs Innovations for Lucent Technologies
Copyright ' 1994 The Australian National University
Copyright ' 1994−2000 Sun Microsystems, Inc.
Copyright ' 1997−2000 Ajuba Solutions.
Copyright ' 1997−2000 Scriptics Corporation.
Copyright ' 1998 Mark Harrison.
Copyright ' 1998−1999 Scriptics Corporation
Copyright ' 1998−2000 Ajuba Solutions
Copyright ' 1999 Scriptics Corporation
Tcl/Tk Keywords – X

ABCDEFGHIJKLMNOPQRSTUVWXYZ

x
  ConfigWind, WindowId
  X window id
  IdToWindow
  xview
  GetScroll

Copyright ' 1989-1994 The Regents of the University of California.
Copyright ' 1990-1994 The Regents of the University of California
Copyright ' 1992-1999 Karl Lehenbauer and Mark Diekhans
Copyright ' 1992-1999 Karl Lehenbauer and Mark Diekhans.
Copyright ' 1993-1997 Bell Labs Innovations for Lucent Technologies
Copyright ' 1994 The Australian National University
Copyright ' 1994-2000 Sun Microsystems, Inc.
Copyright ' 1997-2000 Ajuba Solutions.
Copyright ' 1997-2000 Scriptics Corporation.
Copyright ' 1998 Mark Harrison.
Copyright ' 1998-1999 Scriptics Corporation
Copyright ' 1998-2000 Ajuba Solutions
Copyright ' 1999 Scriptics Corporation
Copyright ' 2000 Jeffrey Hobbs.
Copyright ' 2001 ActiveState Tool Corp.
Copyright ' 2001 Kevin B. Kenny.
Copyright ' 2001 Vincent Darley
Copyright ' 2001-2002 ActiveState Corporation
Copyright ' 2001-2002 Donal K. Fellows
Copyright ' 2001-2002 Kevin B. Kenny. All rights reserved.
Copyright ' 2001-2004 ActiveState Corporation.
Copyright ' 1995-1997 Roger E. Critchlow Jr.
Tcl/Tk Keywords – Y

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

y
  ConfigWind, WindowId
yview
  GetScroll
Tcl/Tk Keywords – Z

Copyright ' 1989−1994 The Regents of the University of California.
Copyright ' 1990−1994 The Regents of the University of California
Copyright ' 1992−1999 Karl Lehenbauer and Mark Diekhans
Copyright ' 1992−1999 Karl Lehenbauer and Mark Diekhans.
Copyright ' 1993−1997 Bell Labs Innovations for Lucent Technologies
Copyright ' 1994 The Australian National University
Copyright ' 1994−2000 Sun Microsystems, Inc.
Copyright ' 1997−2000 Ajuba Solutions.
Copyright ' 1997−2000 Scriptics Corporation.
Copyright ' 1998 Mark Harrison.
Copyright ' 1998−1999 Scriptics Corporation
Copyright ' 1998−2000 Ajuba Solutions
Copyright ' 1999 Scriptics Corporation
Copyright ' 2000 Jeffrey Hobbs.
Copyright ' 2001 ActiveState Tool Corp.
Copyright ' 2001 Kevin B. Kenny.
Copyright ' 2001 Vincent Darley
Copyright ' 2001−2002 ActiveState Corporation
Copyright ' 2001−2002 Donal K. Fellows
Copyright ' 2001−2002 Kevin B. Kenny. All rights reserved.
Copyright ' 2001−2004 ActiveState Corporation.
Copyright ' 1995−1997 Roger E. Critchlow Jr.