

The Tcl Programming Language

A Comprehensive Guide

Second Edition

Ashok P. Nadkarni

The Tcl Programming Language: A Comprehensive Guide

Second Edition

Copyright © 2025 Ashok P. Nadkarni

All rights reserved. No part of this book may be reproduced, stored or transmitted by any means without the prior written permission of the author. If you purchased an electronic version of the book however, you may copy it to multiple devices owned by you. The author makes no warranty, expressed or implied, as to the accuracy of the book contents and assumes no liability for any damages arising from any inaccuracies or errors.

The `asciidoc` and `asciidoc-fop` tools from AsciiDoc project were used to produce the content for this book. Print formatting was done with Apache FOP from the Apache Graphics Project. Text is typeset in the Noto Serif font from Google. Code samples use a combination of Noto Mono from Google and M+ 1p from the M+ Fonts Project.

Table of Contents

Preface	xxv
1. Introduction	1
1.1. A little bit of history	1
1.2. What Tcl offers	1
1.3. Reading this book	2
1.3.1. Typographic conventions	2
1.3.2. Utility procedures used in the book	4
1.4. Online resources	4
2. Getting Started	5
2.1. Installing Tcl	5
2.1.1. Installing with system package managers	5
2.1.2. Binary distributions	6
2.1.3. Building from source	6
2.1.3.1. Tcl source releases	6
2.1.3.2. Build configurations	7
2.1.3.3. Building on Unix-like platforms	7
2.1.3.4. Building on Windows	8
2.1.3.5. Building on macOS	8
2.1.4. Reference documentation	9
2.2. Running a Tcl program	9
2.2.1. The Tcl library and interpreter	9
2.2.2. The tclsh command-line shell	10
2.2.2.1. Running tclsh interactively	10
2.2.2.2. Running programs with tclsh	12
2.2.3. The wish graphical shell	13
2.2.3.1. Running wish interactively	13
2.2.3.2. Running scripts with wish	14
2.2.4. The tkcon enhanced shell	15
2.2.5. Exiting a Tcl application	15
2.2.6. Making Tcl scripts executable	15
2.2.6.1. Executable scripts on Unix	15
2.2.6.2. Executable scripts on Windows	16
2.3. The application runtime environment	17
2.3.1. Command-line arguments	17
2.3.2. The working directory: pwd, cd	18
2.3.3. Environment variables: env	18
2.3.4. The process identifier: pid	19
2.3.5. Executable file path: info nameofexecutable	19
2.3.6. Tcl version information: info tclversion patchlevel	19
2.3.7. Tcl configuration: tcl::pkgconfig, tcl::build-info	20
2.3.8. Platform information	21
3. Tcl Basics	23
3.1. Basic syntax	23
3.2. Substitutions	24
3.2.1. Backslash substitutions	25

3.2.2. Variable substitutions	26
3.2.3. Command substitutions	27
3.3. Quoting	28
3.3.1. Quoting using double quotes	28
3.3.2. Quoting using braces	29
3.3.3. Choosing the quoting mechanism	30
3.4. Argument expansion	31
3.5. Commands	32
3.5.1. Command invocation	32
3.5.1.1. Unknown command handlers	33
3.5.2. Comments	34
3.5.3. Renaming a command	35
3.5.4. Deleting a command	36
3.5.5. Redefining commands	36
3.5.6. Enumerating commands: info commands	37
3.5.7. Command implementation types: info cmdtype	38
3.5.8. Command ensembles	39
3.5.9. Procedures	39
3.5.9.1. Defining procedures: proc	39
3.5.9.2. Procedure parameters	40
3.5.9.2.1. Default argument values	40
3.5.9.2.2. Variable number of arguments	41
3.5.9.2.3. Named parameters and options	41
3.5.9.3. Returning from a procedure: return	43
3.5.9.4. Anonymous procedures: apply	43
3.5.9.5. Introspecting procedures: info procs args default body	45
3.6. Variables	46
3.6.1. Variable name syntax	46
3.6.2. Variable assignment: set	47
3.6.3. Getting a variable's value	47
3.6.4. Unsetting variables: unset	49
3.6.5. Variable scopes, lifetimes and visibility	49
3.6.5.1. Local variables	49
3.6.5.2. Global variables: global	50
3.6.5.3. Creation is not definition	50
3.6.6. Enumerating variables: info vars locals globals	51
3.6.7. Checking variable existence: info exists	52
3.6.8. Array variables	52
3.6.8.1. Basic array operations	52
3.6.8.2. Array defaults: array default	53
3.6.8.3. Checking for arrays: array exists	54
3.6.8.4. Checking for element existence: info exists, array names	54
3.6.8.5. Operating on multiple elements: array set get unset	54
3.6.8.6. Iterating over arrays: array for startsearch nextelement anywhere donesearch	56
3.6.8.7. Array statistics: array size, array statistics	57
3.6.8.8. Printing an array: parray	58
3.6.8.9. More on array keys	58

3.6.9. Constant variables: const	59
3.6.10. Predefined variables	59
3.7. Conditional execution: if	60
3.8. Conditional execution: switch	61
3.9. Looping on a condition: while	63
3.10. Looping over values: for	64
3.11. Terminating loops: break	64
3.12. Skipping loops: continue	65
3.13. Evaluating strings: eval	65
3.13.1. Double substitutions in eval	66
3.14. Evaluating file content: source	68
3.14.1. Retrieving script paths: info script	68
3.15. Introspection	70
3.16. Getting error information	70
3.17. The EIAS principle	71
4. Strings	73
4.1. What is a string	73
4.1.1. Tcl and Unicode	73
4.2. String indices	74
4.3. String literals	75
4.4. Counting characters: string length	75
4.5. Retrieving a character by position: string index	75
4.6. Retrieving substring ranges: string range	75
4.7. Inserting characters: string insert	75
4.8. Appending characters: append	76
4.9. Replace or delete ranges: string replace	76
4.10. Replace or delete substrings: string map	77
4.11. Trimming character sets: string trim trimleft trimright	77
4.12. Concatenating strings: string cat	78
4.13. Joining strings with separators: join	79
4.14. Repeating strings: string repeat	79
4.15. Changing case: string tolower toupper totitle	79
4.16. Reversing a string: string reverse	80
4.17. Searching for substrings: string first last	80
4.18. Searching for word boundaries	80
4.19. Customized interpolation: subst	81
4.20. Formatting strings: format	82
4.20.1. Conversion characters	83
4.20.2. XPG3 format position specifiers	84
4.20.3. Specifying minimum field widths	85
4.20.4. Format flags	85
4.20.5. Precision specifier	86
4.20.6. The size modifier	87
4.21. Parsing strings: scan	87
4.21.1. Conversion characters	89
4.21.2. Scan termination	90
4.21.3. XPG3 scan position specifier	91
4.21.4. Specifying maximum widths	92

4.21.5. The size modifier	92
4.22. Comparing strings: string equal compare	93
4.23. String validation: string is	94
4.24. Glob pattern matching: string match	96
4.25. Matching shared prefixes: ::tcl::prefix	98
5. Lists	101
5.1. Basic list construction: list	101
5.2. List literals	101
5.3. List indices	104
5.3.1. Nested list indices	104
5.4. Retrieving elements by position: lindex	104
5.5. Extracting elements by position: lpop	105
5.6. Retrieving a sublist: lrange	105
5.7. Retrieving leading elements: lassign	105
5.8. Iterating over a list: foreach	106
5.9. Appending elements: lappend	107
5.10. Inserting elements: linsert	107
5.11. Setting element values: lset	108
5.12. Deleting elements: lremove	108
5.13. Replacing elements: lreplace, ledit	109
5.14. Counting elements: llength	110
5.15. Splitting strings into lists: split	110
5.16. Numeric sequences: lseq	111
5.17. Repeating elements: lrepeat	113
5.18. Concatenating lists: concat	113
5.19. Mapping list elements: lmap	113
5.20. Reversing a list: lreverse	114
5.21. Sorting lists: lsort	115
5.21.1. Comparing elements	115
5.21.2. Sort ordering	116
5.21.3. Sorting nested lists with -index	117
5.21.4. Sorting dictionaries with -stride	117
5.21.5. Retrieving sorted indices with -indices	118
5.21.6. Removing duplicate elements	119
5.22. Searching lists: lsearch	120
5.22.1. Search match operators	120
5.22.2. Search operand types	121
5.22.3. Searching nested lists	121
5.22.4. Searching grouped lists	122
5.22.5. Retrieving all matches	122
5.22.6. Retrieving element values	123
5.22.7. Searching sorted lists	123
5.22.8. Specifying a start offset	124
6. Dictionaries	125
6.1. Dictionary literals	125
6.2. Basic dictionary construction: dict create	126
6.3. Nested dictionaries	126
6.4. Dictionary and list compatibility	127

6.5. Checking for a key: dict exists	127
6.6. Retrieving the value for a key: dict get getdef getwithdefault	128
6.7. Enumerating dictionaries: dict keys values	129
6.8. Setting values with dict set	129
6.9. Removing dictionary elements: dict unset remove	130
6.10. Appending to string values: dict append	131
6.11. Appending list elements to values: dict lappend	131
6.12. Incrementing dictionary values: dict incr	132
6.13. Replacing multiple values: dict replace	132
6.14. Combining dictionaries: dict merge	132
6.15. Iterating over dictionaries: dict for	133
6.16. Mapping values: dict map	133
6.17. Filtering dictionaries: dict filter	134
6.18. Shadowing dictionaries with local variables: dict update	135
6.19. Shadowing nested dictionaries: dict with	136
6.20. Count of entries: dict size	138
6.21. Dictionary statistics: dict info	138
6.22. Dictionaries versus arrays	138
7. Numerics	141
7.1. Types and representations	141
7.1.1. The boolean type	141
7.1.2. The integer types	142
7.1.3. The floating point type	142
7.1.3.1. Floating point classification: fpclassify	143
7.1.4. Validation of types	143
7.1.5. Number conversions	144
7.1.5.1. Converting between strings and numbers	144
7.1.5.2. Converting between numeric types	144
7.2. Mathematical operations	145
7.2.1. The tcl::mathop commands	145
7.2.1.1. Arithmetic operator commands	146
7.2.1.2. Comparison operator commands	147
7.2.1.3. String operator commands	148
7.2.1.4. List operator commands	148
7.2.1.5. Bit-wise operator commands	149
7.2.2. Infix expressions: expr	149
7.2.2.1. Comments in expressions	150
7.2.2.2. Operands in expressions	151
7.2.2.3. Operators in expressions	151
7.2.2.4. Grouping operands with parenthesis	153
7.2.2.5. Braces and double substitution	153
7.2.3. Incrementing variables: incr	155
7.2.3.1. Expressions in other commands	155
7.3. Mathematical functions	156
7.3.1. Using functions in expressions	156
7.3.2. Defining custom functions	157
8. Binary data	159
8.1. Binary literals	159

8.2. Encoding binary strings as ASCII	160
8.2.1. Hexadecimal format: binary encode decode hex	160
8.2.2. Base64 format: binary encode decode base64	160
8.2.3. Uuencode format: binary encode decode uuencode	161
8.3. Constructing binary strings: binary format	161
8.3.1. Type specifiers for binary format	163
8.3.2. Cursor movement for formatting	166
8.4. Parsing binary strings: binary scan	166
8.4.1. Type specifiers for binary scan	168
8.4.2. Cursor movement for scanning	171
8.5. Compressing data	172
8.5.1. Compressing strings	172
8.5.1.1. Raw DEFLATE compression: zlib deflate inflate	173
8.5.1.2. Zlib compression: zlib compress decompress	173
8.5.1.3. Gzip compression: zlib gzip gunzip	174
8.5.2. Compressing streams	175
8.5.2.1. Creating a zlib stream: zlib stream	175
8.5.2.2. Writing to a zlib stream	176
8.5.2.3. Finalizing a zlib stream: finalize, put -finalize	176
8.5.2.4. Reading from a zlib stream: get	176
8.5.2.5. Computing zlib stream checksum	177
8.5.2.6. Reusing a zlib stream: reuse	177
8.5.2.7. Closing a zlib stream: close	178
8.5.2.8. Decompression streams	178
8.5.2.9. Flushing zlib streams	178
8.6. Computing checksums: zlib Adler32 CRC32	179
9. Globalization	181
9.1. Character encoding	181
9.1.1. Encoding profiles	182
9.1.2. Supported encodings: encoding names	183
9.1.3. Encoding characters: encoding convertto	183
9.1.4. Decoding characters: encoding convertfrom	184
9.1.5. Adding new encodings: encoding dirs	185
9.1.6. The system encoding: encoding system	185
9.1.7. Reading and writing encoded data	186
9.2. Internationalization	186
9.3. Localization	186
9.3.1. Locales	186
9.3.2. Message catalogs: mcset, mcmset, mcflset, mcflmset	187
9.3.3. Loading message catalogs: mcload, mcloadedlocales	188
9.3.4. Retrieving translations: mc	188
9.3.5. Comparing translation lengths	189
9.3.6. Retrieving and setting the locale: mclocale	189
9.3.7. Locale preferences: mcpreferences	190
9.3.8. Partitioning catalogs with namespaces: mcn	191
9.3.9. Unknown message keys: mcunknown, mcexists	192
9.3.10. Private package locales	193
9.3.10.1. Managing package locales: mcpackagelocale	193

9.3.10.2. Package locale options: mcpackageconfig	193
9.3.10.3. Package namespace: mcpackagenamespaceget	194
9.4. Internationalized Domain Names: tcl::idna	195
10. Regular Expressions	197
10.1. Matching regular expressions: regexp	198
10.1.1. Matching specific characters	198
10.1.2. Matching any character	198
10.1.3. Bracket expressions and character classes	199
10.1.4. Atoms and Groups	201
10.1.5. Quantifiers	202
10.1.6. Alternation and branches	203
10.1.7. Constraints	203
10.1.7.1. Anchoring with ^ and \$	203
10.1.7.2. Constraint escapes	204
10.1.7.3. Lookahead constraints	204
10.1.8. Back references	205
10.1.9. Counting number of matches	206
10.1.10. Retrieving matches	206
10.1.10.1. Retrieving matched content	206
10.1.10.2. Retrieving matched indices	207
10.1.10.3. Retrieving matches with -inline	207
10.1.10.4. Retrieving all matches	207
10.1.11. Option metasyntax	208
10.1.12. Case-independent matching	208
10.1.13. Matching literal strings	208
10.1.14. Newline-sensitive matching	209
10.1.15. Matching at an offset: -start	210
10.1.16. Controlling greediness	210
10.1.17. Comments and expanded syntax	211
10.2. Substituting regular expressions: regsub	212
10.2.1. Computed substitution with regsub	213
11. Dates and Time	215
11.1. POSIX seconds and the epoch	215
11.2. The Julian, Gregorian and alternate calendars	215
11.3. Time zones	216
11.4. Retrieving the current time: clock seconds milliseconds microseconds	217
11.5. Interval measurement: clock clicks	217
11.6. Formatting time for display: clock format	217
11.6.1. Formatting for a different time zone: -timezone, -gmt	218
11.6.2. Formatting for a locale: -locale	218
11.6.3. Controlling display format: -format	218
11.7. Parsing dates and times: clock scan	222
11.7.1. Specifying the parse format: -format	222
11.7.2. Specifying the time zone for parsing: -timezone, -gmt	223
11.7.3. Parsing localized time strings: -locale	223
11.7.4. Validating time strings: -validate	223
11.7.5. Changing the defaults for parsing: -base	224
11.7.6. Free form parsing of time strings	224

11.8. Time arithmetic: clock add	225
11.8.1. Clock computations	225
11.9. Localization	227
11.10. Time representation standards	227
12. Files and File Systems	229
12.1. File paths	229
12.1.1. Path syntax	229
12.1.2. Absolute and relative paths: file pathtype	230
12.1.3. Home directory and tilde substitution	230
12.1.4. Parsing paths: file dirname extension rootname split tail	231
12.1.5. Constructing paths: file join	232
12.1.6. Path normalization: file normalize	232
12.1.7. Converting paths to native form: file nativename	233
12.2. File properties and metadata	234
12.2.1. File size: file size	234
12.2.2. File timestamps: file atime mtime	234
12.2.3. File information: file stat lstat	234
12.2.4. Access checks: file exists readable writable executable owned	235
12.2.5. File types: file isdirectory isfile type	236
12.2.6. File attributes: file attributes	237
12.3. File system operations	239
12.3.1. File system information: file volumes system separator	239
12.3.2. Creating directories: file mkdir	239
12.3.3. Removing files and directories: file delete	240
12.3.4. Copying and renaming: file copy rename	240
12.3.5. Enumerating files: glob	242
12.3.5.1. Matching based on type: -type option	243
12.3.5.2. Changing glob locations: -directory, -path	244
12.3.5.3. Stripping path names: -tails	245
12.3.5.4. Combining path component patterns: -join	245
12.3.5.5. Special considerations for glob	246
12.3.5.5.1. Case sensitivity	246
12.3.5.5.2. Short names on Windows	246
12.3.5.5.3. Enumerating hidden files	246
12.3.6. Links: file link, file readlink	247
12.3.7. Temporary files: file tempfile tempdir	248
13. Channels and Basic I/O	249
13.1. Channels and File I/O	249
13.2. Standard channels: stdin, stdout, stderr	249
13.3. Creating file channels: open	250
13.4. Closing a channel: chan close, close	254
13.5. Channel configuration: chan configure, fconfigure	254
13.6. Writing to channels: chan puts, puts	255
13.6.1. Output buffering	255
13.6.1.1. Buffering mode: -buffering	255
13.6.1.2. Flushing buffers: chan flush, flush	256
13.6.1.3. Sizing buffers: -buffersize	256
13.7. Reading from channels	256

13.7.1. Reading lines from a file: chan gets, gets	257
13.7.2. Reading characters from a file: chan read, read	257
13.7.3. Detecting end of file: chan eof, eof	258
13.7.4. Input buffering	259
13.8. File utilities: writeFile, readFile, foreachLine	259
13.8.1. A utility to write files: writeFile	259
13.8.2. A utility to read files: readFile	259
13.8.3. Iterating over lines: foreachLine	259
13.9. Terminal configuration	260
13.9.1. Input character processing: -inputmode	260
13.9.2. Output screen size: -winsize	260
13.10. Newline translation: -translation	260
13.11. The end of file character: -eofchar	261
13.12. Channel encoding: -encoding	262
13.13. Encoding profiles: -profile	262
13.14. Binary I/O	263
13.15. The file access pointer	264
13.15.1. Retrieving the file access pointer: chan tell, tell	264
13.15.2. Setting the file access position: chan seek, seek	265
13.16. Truncating files: chan truncate	266
13.17. Copying data between channels: chan copy, fcopy	266
13.18. Enumerating open channels: chan names	267
14. Code Execution	269
14.1. Frames and the call stack	269
14.1.1. The call stack	269
14.1.2. Inspecting the call stack: info level	270
14.1.3. Commands that create call frames	272
14.1.4. Referencing variables in call frames: upvar	272
14.1.5. Executing scripts in a call frame: uplevel	276
14.1.6. The internal C stack	280
14.1.7. Recursing in place: tailcall	282
14.1.8. Hidden frames: info frame	286
14.2. Traces	288
14.2.1. Tracing variables: trace add variable	288
14.2.1.1. Tracing array variables	291
14.2.1.2. Applications of variable tracing	293
14.2.2. Tracing commands	296
14.2.2.1. Tracing command lifetimes: trace add command	296
14.2.2.2. Tracing command execution: trace add execution	297
14.2.3. Deleting a trace: trace remove	299
14.2.4. Inspecting traces: trace info	299
14.3. Code construction	300
14.3.1. Scripts versus command prefixes	300
14.3.1.1. Constructing command prefixes	301
14.3.1.2. Constructing scripts	302
14.3.2. Capturing namespace contexts in callbacks	302
14.4. Metaprogramming	302
14.4.1. Procedures with initializers	303

14.4.2. Parsing data	305
14.4.3. Code generalization	308
14.5. Command history: history	309
14.6. Counting command invocations: info cmdcount	312
15. Errors and Exceptions	313
15.1. Dealing with failures	313
15.2. Return codes and the option dictionary	314
15.2.1. Return codes	314
15.2.2. Return code propagation	316
15.2.2.1. Propagating break and continue return codes	316
15.2.2.2. Propagating the return return code	317
15.2.2.3. Propagating the error return code	319
15.2.3. The return options dictionary	319
15.3. The return command	319
15.3.1. Unwinding multiple levels of the call stack	321
15.3.2. Emulating other commands with return	324
15.3.3. Custom return codes	325
15.3.4. Custom return options dictionary	325
15.4. Trapping exceptions	326
15.4.1. Trapping exceptions: catch	326
15.4.2. The error stack and return options dictionary	327
15.4.2.1. Error stack trace: -errorinfo element, errorInfo	327
15.4.2.2. Error line number: -errorline element	327
15.4.2.3. Error codes: -errorcode element, errorCode	328
15.4.2.4. Error stack: -errorstack element, info errorstack	328
15.4.3. Trapping exceptions: try	329
15.5. Raising exceptions	332
15.5.1. Raising errors: throw, error	332
15.5.2. Raising errors: return -code	333
15.6. Forwarding exceptions	334
15.6.1. Forwarding exceptions with return	334
15.6.2. Forwarding exceptions with error	335
15.7. Custom control statements	336
16. Namespaces	339
16.1. Namespace basics	339
16.1.1. A simple namespace example	339
16.1.2. Namespace names and hierarchy	341
16.1.2.1. Inspecting namespace hierarchies: namespace current parent children	343
16.1.2.2. Manipulating names: namespace qualifiers tail	343
16.1.3. Deleting a namespace: namespace delete	344
16.1.4. Checking namespace existence: namespace exists	344
16.2. Executing code in a namespace: namespace eval inscope	345
16.2.1. Namespace contexts in callbacks: namespace code	346
16.3. Namespace variables: variable	347
16.4. Defining commands in a namespace	348
16.4.1. Namespace contexts in procedures	349
16.5. Name resolution	349

16.5.1. Resolving variable names	349
16.5.1.1. Variable resolution outside a procedure	349
16.5.1.2. Variable resolution in a procedure	350
16.5.1.3. Linking to namespace variables: namespace upvar	351
16.5.2. Resolving namespace names	351
16.5.3. Resolving command names	351
16.5.3.1. Importing names: namespace export import forget	352
16.5.3.2. Namespace paths: namespace path	354
16.5.3.3. Comparing namespace imports and paths	355
16.5.3.4. Handling unknown commands: namespace unknown	356
16.5.4. Introspecting name resolution: namespace which origin	357
16.6. Namespace ensembles	359
16.6.1. Ensemble commands	359
16.6.2. Creating ensembles: namespace ensemble create	359
16.6.2.1. Naming an ensemble command	360
16.6.3. Configuring ensembles	361
16.6.3.1. Subcommand configuration: -subcommands, -map	361
16.6.3.2. Subcommand prefixes: option -prefixes	362
16.6.3.3. Subcommand positioning: option -parameters	363
16.6.4. Handling unknown subcommands: option -unknown	364
16.6.5. Checking for ensembles: namespace ensemble exists	366
16.6.6. Nested ensembles	366
16.6.7. Examples of ensembles	367
17. Libraries and Packages	373
17.1. The Tcl system library	373
17.2. Loading libraries on demand: auto_load	374
17.2.1. The tclIndex files: auto_mkindex	374
17.3. Packages	375
17.3.1. Naming packages	375
17.3.2. Package versioning	375
17.3.2.1. Package version syntax	375
17.3.2.2. Comparing package versions: package vcompare vsatisfies	376
17.3.3. Introspecting packages: package names version files	377
17.3.4. Installing packages	378
17.3.5. Searching for libraries	378
17.3.6. Loading packages: package require	378
17.3.6.1. Choosing stable versus unstable packages: package prefer	379
17.3.7. Checking if a package is loaded: package present	380
17.3.8. Registering packages: package ifneeded	380
17.3.9. Creating packages: package provide	381
17.4. Shared library extensions	382
17.4.1. Loading extensions: load	383
17.4.2. Enumerating loaded extensions: info loaded	384
17.5. Modules	384
17.5.1. Module file names	385
17.5.2. Searching for modules	385
17.5.3. Installing modules	387
17.5.4. Creating modules	387

17.6. Packages versus modules	388
17.7. Multiplatform packaging: platform package	388
17.7.1. The platform::shell package	389
17.8. Introspecting package configuration	390
18. Object-Oriented Programming	391
18.1. Objects and classes	391
18.2. Class basics	392
18.2.1. Creating a class	392
18.2.2. Class definition script	393
18.2.3. Destroying classes	394
18.2.4. Data members	394
18.2.4.1. Instance variables: variable, my variable	394
18.2.4.2. Class variables: classvariable	395
18.2.5. Methods	396
18.2.5.1. Constructors and destructors	396
18.2.5.2. Defining methods: method	397
18.2.5.3. Method visibility	397
18.2.5.4. The unknown method	398
18.2.5.5. Class methods: classmethod, myclass	399
18.2.5.6. Deleting methods: deletemethod	400
18.2.5.7. Renaming methods: renamemethod	400
18.2.5.8. Method callbacks: callback, mymethod	401
18.2.5.9. Methods as commands	402
18.2.6. Slot operations	402
18.2.7. Modifying an existing class	403
18.2.8. Class initializer: initialize	403
18.3. Working with objects	403
18.3.1. Creating an object: OBJECT create new	403
18.3.2. Destroying objects	404
18.3.3. Invoking methods	404
18.3.4. Namespace contexts	405
18.3.5. External access to data members: my varname	405
18.4. Inheritance: superclass	406
18.4.1. Methods in derived classes	407
18.4.1.1. Chaining methods	407
18.4.2. Data members in derived classes	408
18.4.3. Multiple inheritance	409
18.4.4. Private contexts	410
18.4.4.1. Defining private contexts: private	411
18.4.4.2. Private methods and forwards	412
18.4.4.3. Private variables	413
18.5. Specializing objects: oo::objdefine	414
18.5.1. Object definition script	414
18.5.2. Object-specific methods	415
18.5.3. Changing an object's class	417
18.6. Using mix-ins	418
18.6.1. Adding a mix-in to a class: mixin	418
18.6.2. Using multiple mix-ins	419

18.6.3. Mix-ins versus inheritance	420
18.7. Method forwarding	420
18.8. Filter methods	422
18.8.1. Defining a filter class	423
18.8.2. When to use filters	424
18.9. Method chains	424
18.9.1. Method chain order	425
18.9.2. Method chain for unknown methods	426
18.9.3. Retrieving the method chain for a class	426
18.9.4. Inspecting method chains within method contexts	427
18.9.5. Looking up the next method in a chain	427
18.9.6. Controlling invocation order of methods	429
18.10. Programming without classes	430
18.11. Metaclasses	431
18.11.1. Implementing a metaclass	433
18.11.2. Abstract classes: <code>oo::abstract</code>	434
18.11.3. Singleton classes: <code>oo::singleton</code>	435
18.11.4. Configurable properties: <code>oo::configurable</code>	436
18.12. OO introspection	438
18.12.1. Enumerating objects and classes	438
18.12.2. Checking if an object is a class	439
18.12.3. Inspecting class relationships	439
18.12.4. Retrieving class definition namespaces	440
18.12.5. Object identity	440
18.12.6. Inspecting an object's class membership	441
18.12.7. Checking if a command is an object	442
18.12.8. Enumerating methods	443
18.12.9. Retrieving method definitions	444
18.12.10. Inspecting method chains and contexts	445
18.12.11. Inspecting filters	445
18.12.12. Enumerating variables	447
18.12.13. Enumerating configurable properties	447
19. The Event Loop	449
19.1. Event sources and types	449
19.2. The Tcl event loop	450
19.2.1. The event and idle task queues	450
19.2.2. Event loop operation	450
19.2.3. Running the event loop	451
19.2.3.1. Processing events based on conditions: <code>vwait</code>	451
19.2.3.1.1. Avoiding deadlocks with <code>vwait</code>	453
19.2.3.2. Single invocation: <code>update</code>	454
19.2.4. Event handlers and the call stack	455
19.3. Scheduling execution of code: <code>after</code>	456
19.3.1. Suspending execution	456
19.3.2. Scheduling code	456
19.3.3. Running on idle: <code>after idle</code>	457
19.3.3.1. Avoiding event queue starvation	458
19.3.4. Cancelling tasks: <code>after cancel</code>	460

19.3.5. Introspecting after handlers: after info	461
19.4. Event loop error handling	461
19.4.1. Custom background error handling: interp bgerror	462
20. Processes and Pipelines	463
20.1. Executing child processes: exec	463
20.1.1. Passing program arguments	464
20.1.2. Locating programs	465
20.1.2.1. Locating shell internal commands: auto_execok	465
20.1.3. Redirecting I/O	466
20.1.3.1. Redirecting input	466
20.1.3.2. Redirecting output	468
20.1.4. Error handling in exec	471
20.1.5. Running background processes	474
20.1.6. Limitations in exec	474
20.2. Channels for process pipelines: open	474
20.2.1. Running tcsh in a pipeline	476
20.2.2. Pipeline process ids: pid	479
20.2.3. Error handling in pipelines	479
20.3. Standalone pipes: chan pipe	479
20.4. Half-closing of channels	483
20.5. Passing environment to child processes	483
20.6. Managing child processes: tcl::process	484
20.6.1. Enumerating subprocesses: tcl::process list	484
20.6.2. Checking child process status: tcl::process status	484
20.6.3. Cleaning up process resources: tcl::process purge autopurge	485
21. Advanced I/O	487
21.1. Asynchronous I/O	487
21.1.1. Non-blocking I/O	488
21.1.1.1. Changing the blocking mode for a channel	488
21.1.1.2. Checking if a channel is blocked	488
21.1.1.3. Non-blocking input	488
21.1.1.3.1. Reading lines in non-blocking mode: chan gets, gets	488
21.1.1.3.2. Reading characters in non-blocking mode: chan read, read	491
21.1.1.4. Non-blocking output: chan puts, puts	492
21.1.2. Event driven I/O: chan event, fileevent	493
21.1.3. Closing non-blocking channels	496
21.1.4. An interactive command line	496
21.2. Channel transforms	498
21.2.1. Channel transform basic operation	498
21.2.2. Implementing channel transforms	499
21.2.2.1. Initializing channel transforms	501
21.2.2.2. Finalizing channel transforms	501
21.2.2.3. Transforming data	502
21.2.2.4. Buffering in channel transforms	503
21.2.2.5. Limiting read-ahead	504
21.2.3. Using channel transforms: chan push pop	504
21.2.4. Zlib channel transforms	505

21.3. Reflected channels	507
21.3.1. Implementing reflected channels	507
21.3.1.1. Initializing a reflected channel	508
21.3.1.2. Closing a reflected channel	509
21.3.1.3. Configuring a reflected channel	509
21.3.1.4. Non-blocking mode and event driven I/O	510
21.3.1.5. Implementing data output	512
21.3.1.6. Implementing data input	514
21.3.1.7. Reflected channel creation: chan create	514
21.3.1.8. Seeking in a reflected channel	515
21.3.1.9. The complete channel implementation	516
21.3.2. Using reflected channels	518
21.3.3. Reflected channel limitations	519
22. Networking and Communications	521
22.1. Network communications	521
22.1.1. Writing TCP clients: socket	521
22.1.1.1. Connecting synchronously	522
22.1.1.2. Connecting asynchronously	522
22.1.2. Writing TCP servers: socket -server	524
22.1.3. Socket configuration	526
22.1.3.1. Text and binary protocols	527
22.1.4. The http package	527
22.2. Communication over serial ports	529
22.2.1. Serial port buffer and queue sizes	529
22.2.2. Serial port speed, parity, and bit lengths	529
22.2.3. Serial port flow control	529
22.2.4. Timers related to serial ports	530
22.2.5. Checking for serial port errors	530
23. Interpreters	531
23.1. Creating interpreters: interp create	531
23.2. Identifying interpreters	532
23.3. Inspecting the interpreter hierarchy: interp children exists	533
23.4. Destroying interpreters	533
23.5. Evaluating scripts in an interpreter: interp eval	533
23.6. Command aliases	534
23.6.1. Defining aliases: interp alias	534
23.6.2. Deleting aliases	536
23.6.3. Introspecting aliases: interp aliases target	536
23.7. Execution context in child interpreters	537
23.8. Cancelling script evaluation: interp cancel	537
23.9. Sharing channels in interpreters	539
23.10. Safe interpreters	540
23.10.1. Creating a safe interpreter	540
23.10.2. Aliasing in safe interpreters	541
23.10.2.1. Precautions for aliased commands	542
23.10.3. Hidden commands	542
23.10.3.1. Invoking hidden commands: interp invokehidden	542
23.10.3.2. Hiding and exposing commands: interp hide expose	544

23.10.3.3. Introspecting hidden commands: <code>interp hidden</code>	545
23.10.4. Trusting safe interpreters: <code>interp marktrusted</code>	545
23.10.5. Safe Tcl	546
23.10.5.1. Creating a Safe Tcl interpreter: <code>safe::interpCreate interpInit</code>	547
23.10.5.2. Deleting a Safe Tcl interpreter: <code>safe::interpDelete</code>	547
23.10.5.3. Configuring a Safe Tcl interpreter: <code>safe::interpConfigure</code>	548
23.10.5.4. Safe Tcl file paths	548
23.10.5.5. Safe Tcl package search	549
23.10.5.6. Troubleshooting Safe Tcl interpreters	549
23.11. Setting resource limits	550
23.11.1. The recursion limit: <code>interp recursionlimit</code>	550
23.11.2. Limiting number of commands: <code>interp limit</code>	550
23.11.3. Limiting interpreter duration: <code>interp limit</code>	552
23.12. Examples using multiple interpreters	553
23.12.1. A safe network server	553
23.12.2. Implementing domain specific languages	555
24. Coroutines	563
24.1. Creating coroutines: <code>coroutine</code>	564
24.2. Suspending and resuming coroutines	564
24.2.1. Yielding to the caller: <code>yield</code>	565
24.2.2. Yielding after initialization	567
24.2.3. Yielding to arbitrary commands: <code>yieldto</code>	568
24.3. Checking coroutine context: <code>info coroutine</code>	569
24.4. Coroutine termination	570
24.4.1. Releasing resources on termination	571
24.5. Exception handling in coroutines	572
24.6. Variable scopes in coroutines	573
24.6.1. Private variables	574
24.7. Coroutines, <code>uplevel</code> and <code>upvar</code>	575
24.8. Coroutines and multiple interpreters	575
24.9. Code injection: <code>coroprobe</code> , <code>coroinject</code>	576
24.10. Using coroutines	578
24.10.1. Explicit and implicit state	578
24.10.2. Generators	579
24.10.3. Emulating objects	582
24.10.4. Producers, consumers, transformers and filters	583
24.10.5. Coroutines and the event loop	586
24.10.6. Emulating blocking calls: <code>coroutine::util</code>	587
24.10.7. Co-operative multitasking	588
25. ZipFS and Single File Deployment	597
25.1. Creating ZIP archives	597
25.1.1. Zipping directories: <code>zipfs mkzip</code>	598
25.1.2. Zipping list of files: <code>zipfs lmkzip</code>	599
25.2. The ZipFS file system	599
25.2.1. The ZipFS root	599
25.2.2. Mounting ZIP archives as a VFS: <code>zipfs mount</code>	599
25.2.3. Mounting in-memory ZIP : <code>zipfs mountdata</code>	600
25.2.4. Introspecting ZipFS mounts: <code>zipfs mount</code>	600

25.2.5. Dismounting ZipFS file systems: zipfs unmount	601
25.2.6. ZipFS utilities: zipfs exists info list find canonical	601
25.2.7. Accessing ZipFS files	602
25.3. Single file deployment	603
25.3.1. Embedded ZIP archives	603
25.3.2. Single-file applications	604
26. Testing and Performance	607
26.1. Testing	607
26.1.1. The tcltest package	607
26.2. Improving performance	611
26.2.1. Profiling scripts	611
26.2.2. Timing scripts	612
26.2.3. The timerate command	612
26.2.4. The time command	614
A. Core packages	615
A.1. The Tk graphical toolkit	615
A.2. The tdbc extension	615
A.3. The http and cookiejar packages	615
A.4. The Thread extension	615
A.5. The tclvfs extension	615
A.6. The registry extension	615
A.7. The dde extension	615
A.8. The IncrTcl extension	616
B. Utility scripts	617
Index	619

List of Figures

2.1. The wish windowing shell	13
2.2. A sample Tk window	14
14.1. Initial call frame	269
14.2. Level 1 call frame	270
14.3. Call stack and upvar	274
14.4. Call stack and uplevel	277
14.5. C stack and call frames	281
14.6. Call stack with tailcall	283
19.1. Call stack in an event handler	455
21.1. Basic channel operation	499

List of Tables

2.1. Configure options	7
2.2. Command-line argument globals	17
2.3. Platform information	21
3.1. Backslash sequences	25
3.2. Command types	38
3.3. Matching options for switch	62
4.1. Integer specifiers for format	83
4.2. String specifiers for format	83
4.3. Floating point specifiers for format	84
4.4. Flag component in format specifiers	86
4.5. Size modifiers for format	87
4.6. Integer specifiers for scan	89
4.7. String specifiers for scan	89
4.8. Integer size modifiers for scan	93
4.9. String validation classes	95
4.10. Pattern matching characters	97
5.1. Lsort comparison options	115
5.2. Lsearch matching options	120
5.3. Lsearch data type options	121
6.1. Differences between tables and arrays	139
7.1. Floating point classes	143
7.2. Arithmetic operators	146
7.3. Comparison operators	147
7.4. String comparison operators	148
7.5. List membership operators	148
7.6. Bit operators	149
7.7. Expression operators in precedence order	152
7.8. Mathematical functions	156
8.1. Type specifiers for binary format	163
8.2. Binary format cursor movement characters	166
8.3. Type specifiers for binary scan	168
8.4. Binary scan cursor movement characters	171
8.5. Gzip header keys	174
8.6. Zlib stream options	175
8.7. Zlib stream put options	176
9.1. Package locale options	194
10.1. Basic regular expression syntax	197
10.2. Regular expression character classes	200
10.3. Character class shorthands	201
10.4. Regular expression quantifiers	202
10.5. Constraint escape sequences	204
11.1. Format groups for clock	219
12.1. File stat array elements	235
12.2. Unix file attributes	237
12.3. Windows file attributes	238

12.4. macOS file attributes	238
12.5. Glob patterns	242
12.6. Glob category 1 type specifiers	243
12.7. Glob category 2 type specifiers	244
13.1. Access modes for open - string form	251
13.2. Access modes for open - list form	251
13.3. Buffering policy option values	256
13.4. Option -inputmode values	260
13.5. Option -translation values	261
13.6. Origin values for seek	265
14.1. Trace operations on variables	289
14.2. Trace operations on commands	296
14.3. Trace operations on command execution	297
15.1. Tcl-defined return codes	315
17.1. Package version requirements syntax	376
18.1. Class definition commands	393
18.2. TclOO slot operations	402
18.3. Object definition commands	415
18.4. oo::configurable property command options	437
19.1. Options controlling vwait conditions	452
19.2. Options for vwait event types	452
20.1. Access mode for pipelines using open	475
21.1. Channel transform subcommands	500
21.2. zlib push command options	506
21.3. Reflected channel subcommands	507
22.1. Socket-specific configuration options	526
22.2. Values for handshake configuration	530
23.1. Safe Tcl predefined aliases	546
23.2. Safe Tcl configuration options	548