

The Tcl Programming Language

A Comprehensive Guide

Ashok P. Nadkarni

The Tcl Programming Language: A Comprehensive Guide

Ashok P. Nadkarni

Copyright © 2017 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 AsciiDoctor 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.

To Aae

Miss you terribly

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
1.5. Chapter summary	5
2. Getting Started	7
2.1. Installing Tcl	7
2.1.1. Installing bundled packages on Linux	7
2.1.2. Installing third party binary distributions	7
2.1.2.1. ActiveState multi-platform distributions	7
2.1.2.2. Installers for Windows	8
2.1.2.3. Perschak distributions for Linux and Windows	9
2.1.2.4. Tcl for Android	9
2.1.2.5. Single file Tcl executables	9
2.1.3. Installing from source	9
2.1.3.1. Tcl source repository and releases	9
2.1.3.2. Building on Unix-like platforms	10
2.1.3.3. Building on Windows	10
2.1.3.4. Building on OS X	11
2.1.3.5. Using BAWT for Tcl and extensions	11
2.1.4. Files and directory structure	11
2.1.5. Reference documentation	11
2.2. Running a Tcl program	12
2.2.1. The Tcl library and interpreter	12
2.2.2. The Tcl shells	12
2.2.2.1. The tclsh command-line shell	12
2.2.2.1.1. Running tclsh interactively	13
2.2.2.1.2. Running scripts with tclsh	15
2.2.2.2. The wish graphical shell	15
2.2.2.2.1. Running wish interactively	15
2.2.2.2.2. Running scripts with wish	17
2.2.2.3. The tkcon enhanced shell	17
2.2.3. Exiting a Tcl application	18
2.2.4. Error messages	18
2.2.5. Making Tcl scripts executable	18
2.2.5.1. Executable scripts on Unix	18
2.2.5.2. Executable scripts on Windows	19
2.3. The application runtime environment	19
2.3.1. Command-line arguments	19
2.3.2. The working directory: pwd, cd	20
2.3.3. Environment variables: env	21
2.3.4. The process identifier: pid	22
2.3.5. Executable file path: info nameofexecutable	22
2.3.6. Tcl version information: info tclversion, info patchlevel	22
2.3.7. Platform information	22
2.3.8. Tcl configuration: tcl::pkgconfig	23
2.4. Chapter summary	24

3. Tcl Basics	25
3.1. Basic syntax	25
3.2. Substitutions	26
3.2.1. Backslash substitutions	26
3.2.2. Variable substitutions	28
3.2.3. Command substitutions	29
3.3. Quoting	29
3.3.1. Quoting using double quotes	30
3.3.2. Quoting using braces	31
3.3.3. Choosing the quoting mechanism	31
3.4. Argument expansion	32
3.5. Commands	33
3.5.1. Command invocation	33
3.5.1.1. Counting command invocations: <code>info cmdcount</code>	34
3.5.1.2. Unknown command handlers	34
3.5.2. Comments	35
3.5.3. Command types	37
3.5.4. Renaming a command	37
3.5.5. Deleting a command	38
3.5.6. Redefining commands	38
3.5.7. Enumerating commands: <code>info commands</code>	39
3.5.8. Procedures	40
3.5.8.1. Defining procedures: <code>proc</code>	40
3.5.8.2. Procedure parameters	40
3.5.8.2.1. Default argument values	41
3.5.8.2.2. Variable number of arguments	41
3.5.8.2.3. Named parameters	42
3.5.8.2.4. Option processing	43
3.5.8.3. Returning from a procedure: <code>return</code>	44
3.5.8.4. Anonymous procedures: <code>apply</code>	45
3.5.8.5. Introspecting procedures: <code>info procs args default body</code>	46
3.6. Variables	48
3.6.1. Variable assignment: <code>set</code>	48
3.6.2. Getting a variable's value	48
3.6.3. Variable name syntax	49
3.6.4. Unsetting variables	50
3.6.5. Variable scopes, lifetimes and visibility	51
3.6.5.1. Local variables	51
3.6.5.2. Global variables: <code>global</code>	51
3.6.5.3. Creation is not definition	52
3.6.6. Variable introspection: <code>info exists vars locals globals</code>	53
3.6.7. Array variables	54
3.6.7.1. Basic array operations	54
3.6.7.2. Printing an array: <code>parray</code>	55
3.6.7.3. Operating on multiple elements: <code>array set</code> , <code>array get</code> , <code>array unset</code>	55
3.6.7.4. Checking for arrays: <code>array exists</code>	56
3.6.7.5. Checking for element existence: <code>info exists</code> , <code>array names</code>	56
3.6.7.6. Array statistics: <code>array size</code> , <code>array statistics</code>	57
3.6.7.7. Iterating over arrays: <code>array startsearch nextelement anymore done</code> <code>search</code>	57
3.6.7.8. More on array keys	58
3.6.8. Predefined variables	59
3.7. Getting error information	59
3.8. Introspection	60

3.9. The EIAS principle	61
3.10. Chapter summary	61
4. Strings	63
4.1. String indices	63
4.2. Constructing strings	63
4.2.1. String literals	63
4.2.2. Concatenating strings: <code>string cat</code>	64
4.2.3. Constructing with substitutions: <code>subst</code>	65
4.2.4. Formatting strings: <code>format</code>	66
4.2.4.1. Conversion characters	67
4.2.4.2. XPG3 format position specifiers	68
4.2.4.3. Specifying minimum field widths	69
4.2.4.4. Format flags	69
4.2.4.5. Precision specifier	70
4.2.4.6. The size modifier	70
4.2.5. Joining strings with separators: <code>join</code>	71
4.2.6. Repeating strings: <code>string repeat</code>	71
4.3. Modifying strings	71
4.3.1. Appending in place: <code>append</code>	71
4.3.2. Replacing substrings by position: <code>string replace</code>	72
4.3.3. Deleting substrings by position	72
4.3.4. Deleting repeated characters at end: <code>string trim trimleft trimright</code>	72
4.4. Comparing strings	73
4.4.1. Comparing for equality: <code>string equal</code>	73
4.4.2. Ordering strings: <code>string compare</code>	73
4.5. Locating and extracting substrings	74
4.5.1. Locating substrings: <code>string first last</code>	74
4.5.2. Retrieving a character by position: <code>string index</code>	74
4.5.3. Retrieving substring ranges: <code>string range</code>	75
4.6. Transforming strings	75
4.6.1. Replacing substrings: <code>string map</code>	75
4.6.2. Changing character case: <code>string tolower, toupper, totitle</code>	76
4.6.3. Reversing a string: <code>string reverse</code>	76
4.6.4. Wrapping text: <code>textutil::adjust, textutil::indent</code>	76
4.7. Parsing strings: <code>scan</code>	77
4.7.1. Scan termination	78
4.7.2. Conversion characters	79
4.7.3. XPG3 scan position specifier	80
4.7.4. Specifying maximum widths	81
4.7.5. The size modifier	81
4.8. Counting characters: <code>string length</code>	82
4.9. String validation: <code>string is</code>	82
4.10. String prefixes: <code>::tcl::prefix</code>	84
4.11. Glob pattern matching	86
4.12. Regular expressions	87
4.12.1. Matching regular expressions: <code>regexp</code>	88
4.12.1.1. Matching specific characters	88
4.12.1.2. Matching any character	89
4.12.1.3. Bracketed expressions and character classes	89
4.12.1.4. Atoms and Quantifiers	91
4.12.1.5. Groups	92
4.12.1.6. Alternation and branches	93
4.12.1.7. Constraints	93
4.12.1.7.1. Anchoring with <code>^</code> and <code>\$</code>	93

4.12.1.7.2. Constraint escapes	93
4.12.1.7.3. Lookahead constraints	94
4.12.1.8. Back references	95
4.12.1.9. Counting number of matches	96
4.12.1.10. Retrieving matches	96
4.12.1.10.1. Retrieving matched content	96
4.12.1.10.2. Retrieving matched indices	96
4.12.1.10.3. Retrieving matches with <code>-inline</code>	97
4.12.1.10.4. Retrieving all matches	97
4.12.1.11. Option metasyntax	97
4.12.1.12. Case-independent matching	98
4.12.1.13. Matching literal strings	98
4.12.1.14. Newline-sensitive matching	99
4.12.1.15. Matching at an offset: <code>-start</code>	99
4.12.1.16. Controlling greediness	100
4.12.1.17. Comments and expanded syntax	101
4.12.2. Substituting regular expressions: <code>regsub</code>	101
4.12.3. Designing regular expressions	102
4.13. Binary strings	103
4.13.1. Binary literals	103
4.13.2. Encoding binary strings as ASCII	103
4.13.2.1. Hexadecimal encoding of binaries: <code>binary encode decode hex</code>	104
4.13.2.2. Base64 format: <code>binary encode decode base64</code>	104
4.13.2.3. Uuencode format: <code>binary encode decode uuencode</code>	104
4.13.3. Constructing binary strings: <code>binary format</code>	105
4.13.3.1. Type specifiers for binary format	106
4.13.3.2. Cursor movement for formatting	109
4.13.4. Parsing binary strings: <code>binary scan</code>	109
4.13.4.1. Type specifiers for binary scan	110
4.13.4.2. Cursor movement for scanning	113
4.14. Character encoding	114
4.14.1. Retrieving supported encodings with encoding names	114
4.14.2. Encoding characters: <code>encode convertto</code>	115
4.14.3. Decoding characters: <code>encode convertfrom</code>	115
4.14.4. Adding new encodings: <code>encoding dirs</code>	115
4.14.5. The system encoding	116
4.14.6. Reading and writing encoded data	116
4.15. Localization and message catalogs	116
4.15.1. Locales	117
4.15.1.1. Retrieving and setting the locale: <code>mclocale</code>	117
4.15.1.2. Locale inheritance: <code>mcpreferences</code>	117
4.15.2. Creating message catalogs: <code>mcset, mcmset, mcflset, mcflmset</code>	118
4.15.3. Loading message catalogs: <code>mclload</code>	119
4.15.4. Retrieving localized strings: <code>mc</code>	119
4.15.5. Partitioning catalogs with namespaces	120
4.15.6. Handling unknown message keys	121
4.16. Data compression	121
4.16.1. Compressing strings	122
4.16.1.1. Raw DEFLATE compression: <code>zlib deflate inflate</code>	122
4.16.1.2. Zlib compression: <code>zlib compress decompress</code>	123
4.16.1.3. Gzip compression: <code>zlib gzip gunzip</code>	123
4.16.2. Compressing streams	124
4.16.2.1. Creating a compression stream	124
4.16.2.2. Writing to a compression stream	125

4.16.2.3. Finalizing a compression stream	125
4.16.2.4. Getting the stream checksum	126
4.16.2.5. Reading from a compression stream	126
4.16.2.6. Reusing a compression stream	126
4.16.2.7. Closing a compression stream	126
4.16.2.8. Decompression streams	126
4.16.2.9. Flushing of compression streams	127
4.16.3. Checksum computation	127
4.16.4. Channel compression transforms	128
4.17. Chapter summary	128
4.18. References	128
5. Lists	129
5.1. Constructing lists	129
5.1.1. List literals	129
5.1.2. Basic list construction: <code>list</code>	131
5.1.3. Splitting strings into lists: <code>split</code>	131
5.1.4. Concatenating lists: <code>concat</code>	133
5.1.5. Repeating elements: <code>lrepeat</code>	133
5.2. List indices	134
5.2.1. Nested list indices	134
5.3. Retrieving elements	134
5.3.1. Retrieving elements by position: <code>lindex</code>	134
5.3.2. Retrieving a list subrange: <code>lrange</code>	135
5.3.3. Retrieving leading elements: <code>lassign</code>	135
5.4. Modifying lists	136
5.4.1. Appending elements: <code>lappend</code>	136
5.4.2. Setting element values: <code>lset</code>	136
5.4.3. Inserting elements: <code>linsert</code>	137
5.4.4. Replacing elements: <code>lreplace</code>	137
5.4.5. Deleting elements	138
5.5. Transforming lists	138
5.5.1. Mapping list elements: <code>lmap</code>	138
5.5.2. Reversing a list: <code>lreverse</code>	139
5.6. Counting elements: <code>llength</code>	139
5.7. Sorting lists: <code>lsort</code>	139
5.7.1. Comparing elements	139
5.7.2. Sort ordering	141
5.7.3. Sorting structured lists	141
5.7.3.1. Sorting nested lists with <code>-index</code>	141
5.7.3.2. Sorting dictionaries with <code>-stride</code>	142
5.7.3.3. Retrieving sorted indices with <code>-indices</code>	142
5.7.4. Removing duplicate elements	143
5.8. Searching lists: <code>lsearch</code>	144
5.8.1. Search match operators	144
5.8.2. Search operand types	145
5.8.3. Searching nested lists	145
5.8.4. Retrieving all matches	146
5.8.5. Retrieving element values	146
5.8.6. Searching sorted lists	146
5.8.7. Specifying a start offset	147
5.9. Iterating over a list: <code>foreach</code>	147
5.10. List utilities	148
5.10.1. Comparing, differencing and merging	149
5.10.2. Permutations and shuffling	150

5.11. Chapter summary	151
6. Dictionaries	153
6.1. Constructing dictionaries	153
6.1.1. Dictionary literals	153
6.1.2. The <code>dict create</code> constructor	154
6.1.3. Creating dictionaries from lists	154
6.1.4. Combining dictionaries: <code>dict merge</code>	154
6.1.5. Nested dictionaries	155
6.2. Reading values from a dictionary	155
6.2.1. Retrieving the value for a key: <code>dict get</code>	155
6.2.2. Enumerating dictionary keys: <code>dict keys</code>	156
6.2.3. Enumerating dictionary values: <code>dict values</code>	156
6.3. Modifying dictionaries	157
6.3.1. Setting values with <code>dict set</code>	157
6.3.2. Removing dictionary elements: <code>dict unset</code>	158
6.3.3. Appending values in-place: <code>dict append</code>	158
6.3.4. Appending list elements to values: <code>dict lappend</code>	159
6.3.5. Incrementing dictionary values: <code>dict incr</code>	159
6.3.6. Removing multiple keys: <code>dict remove</code>	159
6.3.7. Replacing multiple values: <code>dict replace</code>	160
6.3.8. Shadowing dictionaries with local variables: <code>dict update</code>	160
6.3.9. Shadowing nested dictionaries: <code>dict with</code>	161
6.4. Iterating over dictionaries: <code>dict for</code>	162
6.5. Transforming dictionaries	163
6.5.1. Filtering dictionaries: <code>dict filter</code>	163
6.5.2. Mapping values: <code>dict map</code>	164
6.6. Introspecting dictionaries	164
6.6.1. Checking for a key: <code>dict exists</code>	164
6.6.2. Count of entries: <code>dict size</code>	164
6.6.3. Dictionary statistics: <code>dict info</code>	165
6.7. Dictionaries and arrays	165
6.8. Chapter summary	166
7. Numerics	167
7.1. Types and representations	167
7.1.1. The boolean type	167
7.1.2. The integer types	168
7.1.3. The floating point type	169
7.1.4. Validation of types	169
7.1.5. Number conversions	170
7.1.5.1. Converting between strings and numbers	170
7.1.5.2. Converting between numeric types	170
7.2. Mathematical operations	171
7.2.1. The <code>tcl::mathop</code> commands	171
7.2.1.1. Arithmetic operator commands	171
7.2.1.2. Comparison operator commands	172
7.2.1.3. Bit-wise operator commands	173
7.2.1.4. String operator commands	174
7.2.1.5. List operator commands	174
7.2.2. Infix expressions: <code>expr</code>	174
7.2.2.1. Operands in expressions	175
7.2.2.2. Operators in expressions	176
7.2.2.3. Grouping operands with parenthesis	177
7.2.2.4. Braces and double substitution	177
7.2.2.5. Expressions in other commands	179

7.2.3. The <code>incr</code> command	179
7.3. Mathematical functions	180
7.3.1. Using functions in expressions	180
7.3.2. Defining custom functions	181
7.4. Chapter summary	181
8. Dates and Time	183
8.1. Unix time and the epoch	183
8.2. The Julian, Gregorian and alternate calendars	183
8.3. Time zones	184
8.4. Retrieving the current time: <code>clock seconds milliseconds microseconds</code>	184
8.5. Interval measurement	185
8.6. Formatting time for display: <code>clock format</code>	185
8.6.1. Formatting for a different time zone: <code>-timezone, -gmt</code>	185
8.6.2. Formatting for a locale: <code>-locale</code>	185
8.6.3. Controlling display format: <code>-format</code>	186
8.7. Parsing dates and times: <code>clock scan</code>	189
8.7.1. Specifying the parse format: <code>-format</code>	189
8.7.2. Specifying the time zone for parsing: <code>-timezone</code> and <code>-gmt</code>	190
8.7.3. Parsing localized time strings: <code>-locale</code>	190
8.7.4. Changing the defaults for parsing: <code>-base</code>	191
8.7.5. Free form parsing of time strings	191
8.8. Time arithmetic: <code>clock add</code>	192
8.8.1. Clock computations	192
8.9. Time representation standards	193
8.9.1. The ISO 8601 standard	193
8.9.1.1. ISO 8601 calendar dates	194
8.9.1.2. ISO 8601 week dates	194
8.9.1.3. ISO 8601 ordinal dates	195
8.9.1.4. ISO 8601 time	195
8.9.2. RFC 2822 format	196
8.10. Localization	196
8.11. Chapter summary	197
9. Files and Basic I/O	199
9.1. File paths	199
9.1.1. Path syntax	199
9.1.1.1. Absolute and relative paths: <code>file pathtype</code>	200
9.1.1.2. Path normalization: <code>file normalize, fileutil::lexnormalize,</code> <code>fileutil::fullnormalize</code>	200
9.1.1.3. Tilde substitution and the home directory	201
9.1.2. Parsing paths: <code>file split extension rootname dirname tail</code>	202
9.1.3. Constructing paths: <code>file join</code>	203
9.1.4. Converting paths to native form: <code>file nativename</code>	203
9.2. File system operations	203
9.2.1. File system information: <code>file volumes system separator</code>	203
9.2.2. File accessibility: <code>file exists readable writable executable owned</code>	204
9.2.3. File types: <code>file isdirectory isfile type</code>	205
9.2.3.1. File content type: <code>fileutil::fileType</code>	205
9.2.4. File properties	206
9.2.4.1. File size: <code>file size</code>	206
9.2.4.2. File timestamps: <code>file atime mtime</code>	206
9.2.4.3. File information: <code>file stat lstat</code>	206
9.2.4.4. File attributes: <code>file attributes</code>	207
9.2.5. Creating directories: <code>file mkdir</code>	209
9.2.6. Removing files and directories: <code>file delete</code>	209

9.2.7. Copying and renaming: file copy rename	210
9.2.8. Enumerating files: glob	211
9.2.8.1. Matching based on type: -type option	213
9.2.8.2. Changing glob locations: -directory, -path	214
9.2.8.3. Stripping path names: -tails	214
9.2.8.4. Combining path component patterns: -join	215
9.2.8.5. Recursive listing of files	215
9.2.8.6. Special considerations for glob	215
9.2.8.6.1. Case sensitivity	215
9.2.8.6.2. Short names on Windows	215
9.2.8.6.3. Enumerating hidden files	216
9.2.8.6.4. Interaction with tilde expansion	216
9.2.9. Links: file link, file readlink	217
9.2.10. Temporary files: file tempfile, fileutil::tempfile, fileutil::tempdir	218
9.3. Channels and File I/O	219
9.3.1. Standard channels: stdin, stdout, stderr	219
9.3.2. Creating file channels: open	220
9.3.3. Closing a channel: chan close, close	223
9.3.4. Channel configuration: chan configure, fconfigure	223
9.3.5. Writing to channels: chan puts, puts	224
9.3.5.1. Output buffering	225
9.3.5.1.1. Buffering mode: chan configure, fconfigure -buffering	225
9.3.5.1.2. Buffer flushing: chan flush, flush	225
9.3.5.1.3. Controlling the buffer size: chan configure, fconfigure - buffersize	225
9.3.5.2. A wrapper for writing files: fileutil::writeFile	225
9.3.6. Reading from channels	226
9.3.6.1. Reading lines from a file: chan gets, gets	226
9.3.6.2. Reading characters from a file: chan read, read	227
9.3.6.3. Input buffering	227
9.3.6.4. A wrapper for reading files: fileutil::cat	227
9.3.6.5. Iterating over file contents: fileutil::foreachLine	228
9.3.7. Detecting end of file: chan eof, eof	228
9.3.7.1. The end of file character: chan configure, fconfigure -eofchar	228
9.3.8. Channel encoding: chan configure, fconfigure -encoding	229
9.3.9. End of line translation: chan configure, fconfigure -translation	229
9.3.10. Binary I/O	230
9.3.11. The file access pointer	231
9.3.11.1. Retrieving the current file position: chan tell, tell	231
9.3.11.2. Setting the file access position: chan seek, seek	232
9.3.12. Truncating files: chan truncate	233
9.3.13. Copying data between channels: chan copy, fcopy	233
9.3.14. Enumerating open channels: chan names	233
9.3.15. Tcllib fileutil module	234
9.4. Chapter summary	235
10. Code Execution	237
10.1. Evaluating strings: eval	237
10.1.1. Double substitutions in eval	238
10.2. Evaluating file content: source	239
10.3. Conditional execution	241
10.3.1. Evaluating scripts based on an expression: if	241
10.3.2. Evaluating scripts based on patterns: switch	242
10.4. Looping	245
10.4.1. Looping: while	245

10.4.2. Looping: for	246
10.4.3. Loop control	246
10.4.3.1. Terminating loops: break	246
10.4.3.2. Skipping loops: continue	247
10.5. Frames and the call stack	247
10.5.1. The call stack	247
10.5.2. Inspecting the call stack: info level	248
10.5.3. Commands that create call frames	249
10.5.4. Referencing variables in call frames: upvar	250
10.5.5. Executing scripts in a call frame: uplevel	253
10.5.6. The internal C stack	257
10.5.7. Recursing in place: tailcall	258
10.5.8. Hidden frames: info frame	261
10.6. Traces	263
10.6.1. Tracing variables	263
10.6.1.1. Creating a variable trace: trace add variable	263
10.6.1.2. Tracing array variables	269
10.6.1.3. Resource management using variable traces	270
10.6.2. Tracing commands	270
10.6.2.1. Tracing command lifetimes: trace add command	270
10.6.2.2. Tracing command execution: trace add execution	271
10.6.2.3. Deleting a trace: trace remove	272
10.6.2.4. Inspecting traces: trace info	273
10.7. Code construction	274
10.7.1. Scripts versus command prefixes	274
10.7.1.1. Constructing command prefixes	274
10.7.1.2. Constructing scripts	275
10.7.2. Capturing namespace contexts in callbacks	275
10.8. Metaprogramming	275
10.8.1. Procedures with initializers	276
10.8.2. Parsing data by transmutation to code	278
10.8.3. Metaprogramming for specialization	281
10.8.4. Metaprogramming for generalization	282
10.9. Command history: history	284
10.10. Tcl internals	286
10.10.1. How values are stored	286
10.10.1.1. Understanding internal representations	287
10.10.1.2. Data storage and reference counting	289
10.10.1.3. The literal table	290
10.10.2. How code is executed	290
10.10.2.1. Compiler epochs	292
10.10.2.2. The local literals table	292
10.10.2.3. The local variable table	293
10.10.2.4. The stack machine	293
10.10.2.5. Inlined byte code	293
10.10.3. Precompiled byte code: tbcload package	294
10.11. Chapter summary	294
10.12. References	294
11. Errors and Exceptions	295
11.1. Dealing with failures	295
11.2. Return codes and the option dictionary	296
11.2.1. Return codes	296
11.2.2. Return code propagation	297
11.2.3. The return options dictionary	300

11.3. The return command	300
11.3.1. Emulating other commands with return	304
11.3.2. Custom return codes	305
11.3.3. Custom return options dictionary	305
11.4. Trapping exceptions	305
11.4.1. Trapping exceptions: catch	306
11.4.2. The error stack and return options dictionary	306
11.4.2.1. Error stack trace: -errorinfo element, errorInfo	307
11.4.2.2. Error line number: -errorline element	307
11.4.2.3. Error codes: -errorcode element, errorCode	307
11.4.2.4. Error stack: -errorstack element, info errorstack	308
11.4.3. Trapping exceptions: try	308
11.5. Raising exceptions	311
11.5.1. Raising errors: throw, error	311
11.5.2. Raising errors: return -code	312
11.6. Forwarding exceptions	313
11.6.1. Forwarding exceptions: return	313
11.6.2. Forwarding exceptions: error	313
11.7. Custom control statements	314
11.8. Chapter summary	315
11.9. References	315
12. Namespaces	317
12.1. Namespace basics	317
12.1.1. A simple namespace example	317
12.1.2. Namespace names and hierarchy	319
12.1.2.1. Inspecting namespace hierarchies: namespace current, namespace parent, namespace children	320
12.1.2.2. Manipulating names: namespace qualifiers, namespace tail	321
12.1.3. Deleting a namespace: namespace delete	321
12.1.4. Checking namespace existence: namespace exists	322
12.2. Executing code in a namespace: namespace eval inscope	322
12.2.1. Namespace contexts in callbacks	323
12.3. Defining variables in a namespace: variable	324
12.4. Defining commands in a namespace	325
12.4.1. Namespace contexts in procedures	326
12.5. Name resolution	326
12.5.1. Resolving variable names	326
12.5.1.1. Variable resolution outside a procedure	327
12.5.1.2. Variable resolution in a procedure	327
12.5.1.3. Linking to variables in another namespace: namespace upvar	328
12.5.2. Resolving namespace names	328
12.5.3. Resolving command names	329
12.5.3.1. Importing names: namespace export import forget	329
12.5.3.2. Namespace paths: namespace path	331
12.5.3.3. Comparing namespace imports and paths	332
12.5.3.4. Handling unknown commands: namespace unknown	333
12.5.4. Introspecting name resolution: namespace which, namespace origin	334
12.6. Namespace ensembles	335
12.6.1. Ensemble commands	335
12.6.2. Creating a simple ensemble: namespace ensemble create	335
12.6.2.1. Naming an ensemble command	336
12.6.3. Configuring ensembles	337
12.6.3.1. Subcommand configuration: -subcommands, -map	337
12.6.3.2. Subcommand prefixes: option -prefixes	338

12.6.3.3. Subcommand positioning: option -parameters	339
12.6.4. Handling unknown subcommands: option -unknown	339
12.6.5. Checking for ensembles: namespace ensemble exists	341
12.6.6. Nested ensembles	341
12.6.7. Examples of ensembles	342
12.6.7.1. Enhancing existing commands	342
12.6.7.2. Indexing lists by name	343
12.6.7.3. Command objects using ensembles	344
12.7. Chapter summary	346
13. Libraries and Packages	347
13.1. The Tcl system library	347
13.2. Loading libraries on demand: auto_load	348
13.2.1. The tclIndex files	348
13.3. Packages	348
13.3.1. Naming packages	349
13.3.2. Package versioning	349
13.3.2.1. Package version syntax	349
13.3.2.2. Comparing package versions: package vcompare vsatisfies	349
13.3.3. Discovering packages	350
13.3.4. Installing packages	351
13.3.5. Searching for libraries	351
13.3.6. Loading packages: package require	352
13.3.6.1. Choosing stable versus unstable packages	353
13.3.7. Checking if a package is loaded: package present	353
13.3.8. Creating packages	354
13.4. Shared library extensions: load	356
13.4.1. Including shared libraries in packages	357
13.5. Modules	357
13.5.1. Module file names	357
13.5.2. Searching for modules	358
13.5.3. Installing modules	359
13.5.4. Creating modules	360
13.5.5. Including binaries in modules	360
13.6. Packages versus modules	360
13.7. Multiplatform packaging: platform package	361
13.7.1. The platform::shell package	362
13.8. Introspecting package configuration	363
13.9. Chapter summary	363
13.10. References	363
14. Object-Oriented Programming	365
14.1. Objects and classes	365
14.2. Classes	366
14.2.1. Creating a class	366
14.2.2. Destroying classes	367
14.2.3. Defining data members	368
14.2.4. Defining methods	368
14.2.4.1. Method visibility	369
14.2.4.2. Deleting methods	369
14.2.4.3. Renaming methods	369
14.2.5. Constructors and destructors	370
14.2.6. The unknown method	370
14.2.7. Modifying an existing class	371
14.3. Working with objects	371
14.3.1. Creating an object	371

14.3.2. Destroying objects	371
14.3.3. Invoking methods	372
14.3.3.1. Method contexts	372
14.3.4. Accessing data members	372
14.4. Inheritance	373
14.4.1. Methods in derived classes	374
14.4.2. Data members in derived classes	374
14.4.3. Multiple inheritance	375
14.5. Specializing objects	376
14.5.1. Object-specific methods	376
14.5.2. Changing an object's class	377
14.6. Using mix-ins	378
14.6.1. Using multiple mix-ins	379
14.6.2. Mix-ins versus inheritance	380
14.7. Method forwarding	380
14.8. Filter methods	382
14.8.1. Defining a filter class	383
14.8.2. When to use filters	383
14.9. Method chains	384
14.9.1. Method chain order	384
14.9.2. Method chain for unknown methods	385
14.9.3. Retrieving the method chain for a class	385
14.9.4. Inspecting method chains within method contexts	385
14.9.5. Looking up the next method in a chain	386
14.9.6. Controlling invocation order of methods	387
14.10. Programming without classes	388
14.11. OO introspection	389
14.11.1. Introspecting classes	390
14.11.1.1. Enumerating classes	390
14.11.1.2. Checking if an object is a class	390
14.11.1.3. Inspecting class relationships	391
14.11.2. Introspecting objects	391
14.11.2.1. Object identity	391
14.11.2.2. Checking if a command is an object	392
14.11.2.3. Enumerating objects	392
14.11.2.4. Inspecting class membership	392
14.11.3. Introspecting methods	393
14.11.3.1. Enumerating methods	393
14.11.3.2. Retrieving method definitions	394
14.11.3.3. Inspecting method chains and contexts	395
14.11.3.4. Inspecting filters	395
14.11.4. Enumerating data members	396
14.12. Chapter summary	397
14.13. References	397
15. The Event Loop	399
15.1. Event sources and types	399
15.2. The Tcl event loop	399
15.2.1. The event and idle task queues	400
15.2.2. Event loop operation	400
15.2.3. Entering the event loop	400
15.2.3.1. Waiting on a variable: <code>vwait</code>	400
15.2.3.1.1. Avoiding deadlocks with <code>vwait</code>	401
15.2.3.2. Single invocation: <code>update</code>	402
15.2.4. Event handlers and the call stack	403

15.3. Scheduling execution of code: after	404
15.3.1. Suspending execution	404
15.3.2. Scheduling code	404
15.3.3. Running on idle: after idle	405
15.3.3.1. Avoiding event queue starvation	406
15.3.4. Cancelling tasks: after cancel	408
15.3.5. Querying after handlers	408
15.4. Event loop error handling	409
15.4.1. Custom background error handling: interp bgerror	409
15.5. Chapter summary	410
16. Processes and Pipelines	411
16.1. Executing child processes: exec	411
16.1.1. Passing program arguments	412
16.1.2. Locating programs	413
16.1.2.1. Locating internal commands: auto_execok	413
16.1.3. Redirecting I/O	413
16.1.3.1. Redirecting input	413
16.1.3.2. Redirecting output	415
16.1.4. Error handling in exec	417
16.1.5. Running background processes	420
16.1.6. Limitations in exec	420
16.2. Channels for process pipelines: open	421
16.2.1. Running tclsh in a pipeline	423
16.2.2. Pipeline process ids: pid	425
16.2.3. Error handling in pipelines	425
16.3. Standalone pipes: chan pipe	425
16.4. Half-closing of channels	428
16.5. Passing environment to child processes	429
16.6. Interprocess communications	429
16.7. Chapter summary	429
17. Advanced I/O	431
17.1. Asynchronous I/O	431
17.1.1. Non-blocking I/O	432
17.1.1.1. Non-blocking reads	432
17.1.1.1.1. Reading lines in non-blocking mode: chan gets, gets	432
17.1.1.1.2. Reading characters in non-blocking mode: chan read, read	434
17.1.1.2. Non-blocking writes: chan puts, puts	436
17.1.2. Event driven I/O: chan event, filevent	436
17.1.3. Closing non-blocking channels	438
17.1.4. An interactive command line	439
17.2. Channel transforms	440
17.2.1. Channel transform basic operation	440
17.2.2. Implementing channel transforms	442
17.2.2.1. Initializing channel transforms	443
17.2.2.2. Finalizing channel transforms	443
17.2.2.3. Transforming data	444
17.2.2.4. Buffering in channel transforms	444
17.2.2.5. Limiting read-ahead	445
17.2.3. Using channel transforms	445
17.2.4. Applications of channel transforms	447
17.2.5. Zlib channel transforms	447
17.3. Reflected channels	449
17.3.1. Implementing reflected channels	449
17.3.1.1. Initializing a reflected channel	450

17.3.1.2. Closing a reflected channel	450
17.3.1.3. Configuring a reflected channel	450
17.3.1.4. Non-blocking mode and event driven I/O	451
17.3.1.5. Implementing data output	453
17.3.1.6. Implementing data input	455
17.3.1.7. Reflected channel creation	455
17.3.1.8. Seeking in a reflected channel	456
17.3.1.9. The complete channel implementation	456
17.3.2. Using reflected channels	459
17.3.3. Reflected channel limitations	459
17.3.4. Applications of reflected channels	460
17.4. Chapter summary	461
17.5. References	461
18. Networking and Communications	463
18.1. Network communications	463
18.1.1. IP addresses	463
18.1.2. DNS names	464
18.1.3. Text and binary protocols	465
18.1.4. Communicating over TCP	465
18.1.4.1. Writing TCP clients: <code>socket</code>	466
18.1.4.1.1. Connecting synchronously	466
18.1.4.1.2. Connecting asynchronously	467
18.1.4.2. Writing TCP servers: <code>socket -server</code>	468
18.1.4.3. TCP connection state	470
18.1.5. Communicating over UDP	470
18.1.5.1. Creating a UDP socket	471
18.1.5.2. Sending UDP datagrams	471
18.1.5.3. Receiving UDP datagrams	472
18.1.5.4. Receiving broadcast datagrams	472
18.1.5.5. Multicast operation	473
18.1.5.6. Best practices for UDP	473
18.1.6. Layered protocols	474
18.2. Communication over serial ports	474
18.2.1. Serial port speed, parity, and bit lengths	475
18.2.2. Serial port flow control	475
18.2.3. Serial port buffer and queue sizes	475
18.2.4. Timers related to serial ports	476
18.2.5. Checking for serial port errors	476
18.3. Chapter summary	476
18.4. References	476
19. Virtual File Systems and Tkits	477
19.1. Using VFS	477
19.1.1. URL mounts	479
19.2. Implementing a VFS	480
19.2.1. Signalling VFS errors	481
19.2.2. Mounting and unmounting	481
19.2.3. VFS operations	482
19.2.3.1. Checking access: <code>access</code>	482
19.2.3.2. Creating directories: <code>createdirectory</code>	483
19.2.3.3. Deleting directories: <code>removedirectory</code>	483
19.2.3.4. Creating and opening files: <code>open</code>	483
19.2.3.5. Deleting files: <code>deletefile</code>	484
19.2.3.6. Setting file timestamps: <code>utime</code>	484
19.2.3.7. File statistics: <code>stat</code>	485

19.2.3.8. File attributes: fileattributes	485
19.2.3.9. Matching files: matchindirectory	486
19.2.3.10. memfs internals	487
19.3. VFS introspection	488
19.4. Single file deployment: Tclkit, Starkit, Starpack	488
19.4.1. Tclkits, starkits, starpacks	489
19.4.2. Obtaining a tclkit	489
19.4.2.1. Downloading prebuilt tclkits	489
19.4.2.2. Building tclkits	489
19.4.3. Using tclkits as Tcl shells	490
19.4.4. The sdx tool	490
19.4.5. Building a single script starkit: sdx qwrap	491
19.4.6. Building a single script executable: sdx qwrap -runtime	492
19.4.7. Internal structure of a starkit	492
19.4.8. A more complete starkit example: sdx wrap	494
19.4.9. Considerations for multiplatform starkits	498
19.4.10. Starkit mount points	499
19.4.11. Writable starkits	500
19.5. Chapter summary	500
19.6. References	500
20. Interpreters	501
20.1. Interpreter basics	501
20.1.1. Creating interpreters: interp create	501
20.1.2. Identifying interpreters	502
20.1.3. Inspecting the interpreter hierarchy	503
20.1.4. Destroying interpreters	503
20.2. Evaluating scripts in an interpreter	503
20.3. Command aliases	504
20.3.1. Defining aliases	504
20.3.2. Introspecting aliases	505
20.4. Execution context in slaves	506
20.5. Cancelling script evaluation	506
20.6. Safe interpreters	508
20.6.1. Creating a safe interpreter	509
20.6.2. Aliasing in safe interpreters	509
20.6.2.1. Precautions for aliased commands	510
20.6.3. Hidden commands	510
20.6.3.1. Invoking hidden commands	510
20.6.3.2. Hiding and exposing commands	512
20.6.3.3. Introspecting hidden commands	513
20.6.4. Trusting safe interpreters	513
20.6.5. Utilities for safe interpreters	513
20.6.5.1. Safe Tcl	514
20.6.5.2. The island package	516
20.7. I/O in slave interps	517
20.8. Setting resource limits	517
20.8.1. The recursion limit	518
20.8.2. Limiting interpreter lifetimes	518
20.8.2.1. Limiting number of commands executed	519
20.8.2.2. Limiting interpreter duration	520
20.9. Using multiple interpreters	521
20.9.1. A safe network server	521
20.9.2. Implementing domain specific languages	522
20.10. Chapter summary	528

21. Coroutines	529
21.1. Creating coroutines: <code>coroutine</code>	529
21.2. Suspending and resuming coroutines	530
21.2.1. Yielding to the caller: <code>yield</code>	530
21.2.2. Yielding after initialization	532
21.2.3. Yielding to arbitrary commands: <code>yieldto</code>	533
21.3. Checking coroutine context: <code>info coroutine</code>	534
21.4. Coroutine termination	534
21.4.1. Releasing resources on termination	536
21.5. Exception handling in coroutines	536
21.6. Variable scopes in coroutines	537
21.6.1. Private variables	538
21.7. Coroutines, <code>uplevel</code> and <code>upvar</code>	539
21.8. Coroutines and multiple interpreters	539
21.9. Code injection	539
21.10. Using coroutines	541
21.10.1. Explicit and implicit state	541
21.10.2. Generators	542
21.10.2.1. The generator package	544
21.10.3. Emulating objects	546
21.10.4. Producers, consumers, transformers and filters	547
21.10.4.1. Coroutines and the event loop	550
21.10.5. Emulating blocking calls: <code>coroutine::util</code>	550
21.10.6. Co-operative multitasking	552
21.11. Chapter summary	558
21.12. References	558
22. Threads	559
22.1. Enabling thread support: the <code>Thread</code> package	559
22.2. Threading model	560
22.3. Creating threads: <code>thread::create</code>	561
22.4. Interthread communication	562
22.4.1. Waiting for messages: <code>thread::wait</code>	562
22.4.1.1. Limiting message queue size: <code>thread::configure -eventmark</code>	562
22.4.2. Sending messages: <code>thread::send</code>	563
22.4.3. Broadcasting messages: <code>thread::broadcast</code>	563
22.5. Thread lifetime management	564
22.5.1. Waiting for thread completion: <code>thread::join</code>	564
22.5.2. Thread reference counting: <code>thread::preserve</code> , <code>thread::release</code>	565
22.6. Canceling script execution: <code>thread::cancel</code>	566
22.7. Error handling in threads: <code>thread::errorproc</code>	566
22.8. Threads and I/O channels: <code>thread::transfer</code>	568
22.9. Introspecting threads: <code>thread::id</code> , <code>names</code> , <code>exists</code>	570
22.10. Thread-shared variables	570
22.10.1. Scalar operations on shared variables	571
22.10.2. List operations	572
22.10.3. Array operations	573
22.10.4. Keyed lists	574
22.10.5. Introspection and utility commands: <code>thread::names</code> , <code>object</code> , <code>lock</code>	575
22.11. Synchronization and locking	576
22.11.1. Mutexes: <code>thread::mutex</code> , <code>thread::rwmutex</code>	577
22.11.2. Condition variables: <code>thread::cond</code>	579
22.12. Thread pools	581
22.12.1. Creating a thread pool: <code>tpool::create</code>	581
22.12.2. Posting jobs to a thread pool: <code>tpool::post</code> , <code>tpool::get</code>	583

22.12.3. Waiting for job completion: <code>tpool::wait</code>	583
22.12.4. Retrieving a job result: <code>tpool::get</code>	584
22.12.5. Canceling a job: <code>tpool::cancel</code>	584
22.12.6. Suspending thread pools: <code>tpool::suspend</code> , <code>tpool::resume</code>	585
22.12.7. Thread pool lifetimes: <code>tpool::preserve</code> , <code>tpool::release</code>	585
22.13. Distributing interpreter state: the <code>Ttrace</code> package	585
22.14. Using extensions in threads	586
22.15. Comparing coroutines and threads	587
22.16. Chapter summary	587
22.17. References	588
23. Database Connectivity with TDBC	589
23.1. Installing TDBC	589
23.2. Loading TDBC	589
23.3. Concepts	590
23.4. Connecting to databases	590
23.4.1. Connecting to SQLite: <code>tdbc::sqlite3::connection</code>	590
23.4.2. Connecting via ODBC: <code>tdbc::odbc::connection</code>	591
23.4.3. Connecting to MySQL: <code>tdbc::mysql::connection</code>	592
23.4.4. Connecting to PostgreSQL: <code>tdbc::postgres::connection</code>	593
23.4.5. Common connection options	593
23.4.6. Configuring connections: <code>DBCINN configure</code>	594
23.4.7. Releasing connection resources: <code>DBCINN close</code>	594
23.5. Executing SQL	594
23.5.1. Preparing a statement: <code>DBCINN prepare</code>	594
23.5.2. Executing a prepared statement: <code>STMT execute</code>	594
23.5.3. Bound variables	595
23.5.3.1. Bound variable configuration: <code>STMT paramtype</code>	596
23.5.4. Closing prepared statements: <code>STMT close</code>	596
23.5.5. Direct evaluation (MySQL only): <code>DBCINN evaldirect</code>	596
23.6. Retrieving data from result sets	597
23.6.1. Introspecting result sets: <code>RESULTSET columns</code>	597
23.6.2. Retrieving result set rows: <code>RESULTSET nextlist nextdict nextrow</code>	597
23.6.3. Multiple result sets: <code>RESULTSET nextresults</code>	597
23.6.4. Releasing result sets: <code>RESULTSET close</code>	598
23.6.5. Convenience wrappers for retrieval	598
23.6.5.1. Retrieving a complete result set: <code>allows</code>	598
23.6.5.2. Iterating over result sets: <code>foreach</code>	601
23.7. Database transactions	601
23.7.1. Using the transaction method	601
23.7.2. Using <code>begintransaction</code> , <code>commit</code> and <code>rollback</code>	602
23.8. Handling NULL values	603
23.9. Stored procedures: <code>DBCINN preparecall</code>	603
23.10. Introspection	604
23.10.1. Enumerating objects: <code>DBCINN statements</code>	604
23.10.2. Introspecting tables: <code>DBCINN tables</code>	604
23.10.3. Introspecting columns: <code>DBCINN columns</code>	604
23.10.4. Introspecting keys: <code>DBCINN primarykeys foreignkeys</code>	605
23.11. ODBC utilities	606
23.12. Chapter summary	607
23.13. References	607
24. Testing and Performance	609
24.1. Testing	609
24.1.1. The <code>tcltest</code> package	609
24.1.2. Testing interactive applications	612

24.1.3. Source code checkers	613
24.2. Improving performance	613
24.2.1. Profiling scripts	613
24.2.2. Timing scripts	614
24.2.3. Performance hints	616
24.3. Chapter summary	619
A. Libraries and Extensions	621
A.1. GUI toolkits	621
A.2. Internet protocols	621
A.3. Web servers and frameworks	622
A.4. Numeric computing	622
A.5. Database access	622
A.6. XML processing	623
A.7. Integration with other languages	623
A.8. Image processing	623
A.9. Platform-specific extensions	624
A.9.1. Windows extensions	624
A.9.2. Unix extensions	624
A.9.3. Android extensions	624
B. Utility scripts	625
Index	627

List of Figures

2.1. The wish windowing shell	16
2.2. A sample Tk window	16
10.1. Initial call frame	247
10.2. Level 1 call frame	248
10.3. Call stack and upvar	251
10.4. Call stack and uplevel	254
10.5. C stack and call frames	258
10.6. Call stack with tailcall	259
15.1. Call stack in an event handler	403
17.1. Basic channel operation	441

List of Tables

2.1. Tcl directory structure	11
2.2. Command-line argument globals	20
2.3. Platform information	22
3.1. Backslash sequences	27
4.1. String specifiers for format	67
4.2. Integer specifiers for format	67
4.3. Floating point specifiers for format	67
4.4. String specifiers for scan	79
4.5. Integer specifiers for scan	80
4.6. Integer size modifiers for scan	81
4.7. String validation classes	83
4.8. Pattern matching characters	86
4.9. Basic regular expression syntax	88
4.10. Regular expression character classes	90
4.11. Character class shorthands	91
4.12. Regular expression quantifiers	92
4.13. Constraint escape sequences	94
4.14. Table	99
4.15. Type specifiers for binary format	106
4.16. Binary format cursor movement characters	109
4.17. Type specifiers for binary scan	111
4.18. Binary scan cursor movement characters	113
4.19. Gzip header keys	123
4.20. Compression stream options	125
4.21. Compression stream put options	125
5.1. Lsort comparison options	139
5.2. Lsearch matching options	144
5.3. Lsearch data type options	145
6.1. Differences between tables and arrays	165
7.1. Arithmetic operators	171
7.2. Comparison operators	172
7.3. Bit operators	173
7.4. Expression operators in precedence order	176
7.5. Mathematical functions	180
8.1. Format groups for clock	186
9.1. File stat array elements	207
9.2. Windows file attributes	208
9.3. Unix file attributes	209
9.4. Mac OS X file attributes	209
9.5. Glob patterns	212
9.6. Glob category 1 type specifiers	213
9.7. Glob category 2 type specifiers	213
9.8. Access mode for open - string form	220
9.9. Access mode for open - list form	222
9.10. Buffering policy option values	225
9.11. Option -translation values	230
9.12. Origin values for seek	232
10.1. Matching options for switch	244
10.2. Trace operations on variables	264
10.3. Trace operations on commands	270
10.4. Trace operations on command execution	271

10.5. Disassembly header	292
11.1. Tcl-defined return codes	297
13.1. Package version requirements syntax	350
16.1. Access mode for pipelines using open	422
17.1. Channel transform subcommands	442
17.2. zlib push command options	448
17.3. Reflected channel subcommands	449
18.1. Socket-specific configuration options	470
18.2. Values for handshake configuration	475
19.1. VFS driver subcommands	480
19.2. Starkit start-up modes	496
20.1. Safe Tcl predefined aliases	514
20.2. Safe Tcl configuration options	515
22.1. Thread package namespaces	560
22.2. Thread pool options	582
23.1. Core TDBC driver packages	590
23.2. MySQL connection options	592
23.3. PostgreSQL connection options	593
23.4. Connection options common to all TDBC drivers	593
23.5. Column description keys	605
23.6. Foreign key dictionary	606
23.7. tdbc::odbc::datasource commands	606

Preface

...or why I wrote this book.

This book, as the more perceptive readers would have deduced from the title, is about the Tcl programming language.

About Tcl

Tcl was one of the first “dynamic” languages to become popular, seeing widespread use beginning with the early 90’s. Along with its accompanying graphical programming toolkit Tk, the language was influential enough for John Ousterhout, its inventor, to be conferred the ACM Software System Award in 1998. Since that time, Tcl has found its way into every application category you can imagine¹ and Tcl deployments run the gamut from embedded devices like routers to Internet facing, galaxy class servers to distributed back-end infrastructure.

Yet, despite its wide use and adoption, Tcl has not gained the notoriety of the newer languages that have sprung up in the past few years. In large part this is because the Tcl community as a whole has never been particularly interested in evangelizing the language. As a result, people’s perception of Tcl has remained stuck in the last century in terms of features and capabilities.

This book hopes to remedy that by providing comprehensive coverage of the current version, 8.6 as of writing, of Tcl. It aims to be a tutorial, programming guide and reference for the language and its runtime environment.

About the book

I have tried to cover every single feature of the Tcl programming language and core libraries in the book. However, space limitations do not permit inclusion of the myriad third-party libraries and extensions to Tcl. Important and commonly used ones are touched upon briefly so you are made aware of their existence but they are not described in detail. Even the graphical toolkit Tk, which is associated so strongly with Tcl that most people refer to the two in conjunction as Tcl/Tk, is not included. It would double the size of the book by itself. For the same reason, the book also does not discuss the C programming interface to Tcl although ease of interfacing to C is one of the strengths of the language.

About me

Give me a place to stand and I’ll move the earth.

— Archimedes *famous Greek mathematician*

Give me Tcl and I’ll ship on time.

— *Me famous Tcl author*

Authoring is hard work. Now a *fainéant* I anient² so I don’t mind hard work. But it is **tedious** hard work (Oh Lord, indexing!) which is a whole different kettle of fish. And there is not much money to be made in technical books, it being difficult to insert violence and gratuitous sex into the prose to sell a few more copies. So what was my motivation in writing this book? Likely the same as most authors of technical books — when you believe in a language or technology, you have an urge, a compulsion, to spread the word.

My association with Tcl includes intermittent stretches over a couple of decades. I have run engineering in start-ups where significant components of the product were developed in Tcl. I have also authored several open source extensions and libraries for Tcl. In that time I grew to be increasingly enamoured of the language for one primary reason — productivity. From the perspective of an individual programmer, this productivity results from Tcl’s malleability and rich feature set that facilitate a number of different programming styles and architectural patterns. From a management perspective, Tcl is stable, portable and versatile enough to be of benefit throughout a product’s life-cycle — from development, test, deployment to field support.

¹ See <http://wiki.tcl.tk/37889>.

² With apologies to Ogden Nash

My hope is that this book will in some small way help popularize Tcl — beat the drum, so to speak, with regard to its simplicity and power.

Acknowledgements

First up is of course Professor John Ousterhout without whom there would be no Tcl. Beyond that, I thank en masse

- the members of the Tcl Core Team (TCT) that now directs development and future direction of the language
- the contributors to the TcLer's Wiki³, which contains a trove of illustrative examples and expository material about Tcl, and the Tcl Chat room where much illuminating discussion takes place
- the many individuals who are collectively responsible for the libraries and tools that complete the Tcl ecosystem.

I would also like to thank Arjen Markus and Doran Moppert who reviewed portions of the book with patience and an eagle eye.

In terms of the book itself, a shout out to the AsciiDoctor⁴ team whose open source publishing toolchain greatly eased the actual process of writing and producing both electronic and print formats. Outstanding software.

Contacting me

You may contact me via my website <http://www.magic splat.com> which also happens to be the home of my Tcl-related blog, articles and software. Alternatively, you may reach me through the TcLer's Wiki or the `comp.lang.tcl` newsgroup, both of which I follow regularly.

Ashok P. Nadkarni
Bengaluru, May 2017

³ <http://wiki.tcl.tk>

⁴ <http://www.asciidoctor.org>