Tcl and the Tk Toolkit

Second Edition

John K. Ousterhout • Ken Jones
With contributions by Eric Foster-Johnson,
Donal Fellows, Brian Griffin, and David Welton

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

Excerpts from the Tcl/Tk reference documentation are used under the terms of the Tcl/Tk license (http://www.tcl.tk/software/tcltk/license.html).

The open source icon set used in Figures 22-1, 22-2, and 22-3 are from the Tango Desktop Project (http://tango.freedesktop.org/Tango_Desktop_Project).

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Ousterhout, John K.
Tcl and the Tk toolkit / John Ousterhout, Ken Jones ; with contributions by Eric Foster-Johnson . . . [et al.]. — 2nd ed.
p. cm.
Includes index.
ISBN 978-0-321-33633-0 (pbk. : alk. paper)
1. Tcl (Computer program language) 2. Tk toolkit. I. Jones, Ken. II. Title.
QA76.73.T44O97 2009
005.13'3—dc22

2009022005

Copyright © 2010 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax: (617) 671-3447

ISBN-13: 978-0-321-33633-0
ISBN-10: 0-321-33633-X
Text printed in the United States on recycled paper at Edwards Brothers in Ann Arbor, Michigan.
First printing, August 2009

Preface

Tcl and Tk have evolved tremendously since John wrote the first edition of this book. In a "History of Tcl" that John wrote several years ago (`http://www.tcl.tk/about/history.html`), he documented several significant enhancements that occurred:

> I joined Sun in May of 1994 and began building a team of Tcl developers . . . The additional resources provided by Sun allowed us to make major improvements to Tcl and Tk. Scott Stanton and Ray Johnson ported Tcl and Tk to Windows and the Macintosh, so that Tcl became an outstanding cross-platform development environment . . . Jacob Levy and Scott Stanton overhauled the I/O system and added socket support, so that Tcl could easily be used for a variety of network applications. Brian Lewis built a bytecode compiler for Tcl scripts, which provided speed-ups of as much as a factor of 10x. Jacob Levy implemented Safe-Tcl, a powerful security model that allows untrusted scripts to be evaluated safely. Jacob Levy and Laurent Demailly built a Tcl plugin, so that Tcl scripts can be evaluated in a web browser, and we created Jacl and TclBlend, which allow Tcl and Java to work closely together. We added many other smaller improvements, such as dynamic loading, namespaces, time and date support, binary I/O, additional file manipulation commands, and an improved font mechanism.

John went on to found the company Scriptics in 1997 to create development tools and provide training and support for Tcl on a commercial basis. I joined Scriptics shortly after its founding and had the pleasure of working with John and many other talented people responsible for the success of Tcl and Tk. During that time, Tcl became the first dynamic language with native Unicode support (for internationalization), thread safety (for multithreaded applications), and an all-new regular expression package by Henry Spencer that included many new features as well as Unicode support. In 1998, John was recognized for his work on Tcl/Tk with the ACM Software System Award, conferred each year for "a software system that has had a lasting influence."

After many years of serving as the "benevolent dictator" of Tcl/Tk development, John was ready to focus on other ventures. In 2000, he turned control of Tcl/Tk over to the Tcl Core Team (TCT), a group of Tcl experts who collectively manage the development of Tcl/Tk. The TCT organizes its activities with Tcl Improvement Proposals (TIPs). Each TIP is a short document that describes a particular project, activity, or process. Anyone can write a TIP; the TIP then gets circulated among the Tcl Core Team for discussion and approval. You can learn more about the process at `http://www.tcl.tk/community/coreteam`. The TCT welcomes community involvement in shaping the future of Tcl/Tk.

One of the most exciting developments in recent years is a technology supporting single-file distribution of Tcl runtime environments and Tcl/Tk-based applications through *Starkits* and *Starpacks*; Chapter 14 shows you how to take advantage of this method of distributing your applications. Starkits are based on another powerful innovation, *virtual file systems*, which allows your applications to treat such entities as ZIP archives, FTP sites, HTTP sites, and WebDAV shares as mountable file systems; Chapter 11 describes using this technology. And Tk 8.5 introduced a new set of *themed widgets*, which provide a more modern appearance and consistent look and feel than the classic Tk widgets; Chapter 19 introduces the capabilities of the new themed widgets.

In addition to my coauthors, many other people have contributed to the success of this second edition. Clif Flynt, Jeff Hobbs, Brian Kernighan, Steve Landers, and Mark Roseman all spent significant time and effort on critiquing and improving the technical content. Joe English, Jeff Hobbs, and Don Porter provided valuable insight into several nooks and crannies of Tcl and Tk. Mark Roseman and his TkDocs site (`http://www.tkdocs.com`) provided invaluable insight into themed widgets, styles, and themes. Cameron Laird served as a sounding board for the initial shaping and restructuring of the second edition. At Addison-Wesley, long-suffering Mark Taub and Debra Williams Cauley kept working with me to bring this book to publication, and Michael Thurston helped make my inconsistencies more consistent. Finally, Dean Akamine spent many a long hour on the thankless job of converting files from format to format and helping me to tame FrameMaker. To them and all the others I've negligently forgotten to name, I give my thanks for their help.

Ken Jones
San Francisco, California
April 2009

Preface to the First Edition

Tcl was born of frustration. In the early 1980s my students and I developed a number of interactive tools at the University of California at Berkeley, mostly for integrated circuit design, and we found ourselves spending a lot of time building bad command languages. Each tool needed to have a command language of some sort, but our main interest was in the tool rather than its command language. We spent as little time as possible on the command language and always ended up with a language that was weak and quirky. Furthermore, the command language for one tool was never quite right for the next tool, so we ended up building a new bad command language for each tool. This became increasingly frustrating.

In the fall of 1987 it occurred to me that the solution was to build a reusable command language. If a general-purpose scripting language could be built as a C library package, then perhaps it could be reused for many different purposes in many different applications. Of course, the language would need to be extensible so that each application could add its own specific features to the core provided by the library. In the spring of 1988 I decided to implement such a language, and the result was Tcl.

Tk was also born of frustration. The basic idea for Tk arose in response to Apple's announcement of HyperCard in the fall of 1987. HyperCard generated tremendous excitement because of the power of the system and the way in which it allowed many different interactive elements to be scripted and work together. However, I was discouraged. The HyperCard system had obviously taken a large development effort, and it seemed unlikely to me that a small group such as a university research project could ever mount such a massive effort. This suggested that we would not be able to participate in the development of new forms of interactive software in the future.

I concluded that the only hope for us was a component approach. Rather than building a new application as a self-contained monolith with hundreds of thousands of lines of code, we needed to find a way to divide applications into many smaller reusable components. Ideally each component would be small enough to be implemented by a small group, and interesting applications could be created by assembling components. In this environment it should be possible to create an exciting new application by developing one new component and then combining it with existing components.

The component-based approach requires a powerful and flexible "glue" for assembling the components, and it occurred to me that perhaps a shared scripting language could provide that glue. Out of this thinking grew Tk, an X11 toolkit based on Tcl. Tk allows components to be either individual user-interface controls or entire applications; in either case components can be developed independently and Tcl can be used to assemble the components and communicate between them.

I started writing Tcl and Tk as a hobby in my spare time. As other people began to use the systems I found myself spending more and more time on them, to the point where today they occupy almost all of my waking hours and many of my sleeping ones.

Tcl and Tk have succeeded beyond my wildest dreams. The Tcl/Tk developer community now numbers in the tens of thousands and there are thousands of Tcl applications in existence or under development. The application areas for Tcl and Tk cover virtually the entire spectrum of graphical and engineering applications, including computer-aided design, software development, testing, instrument control, scientific visualization, and multimedia. Tcl is used by itself in many applications, and Tcl and Tk are used together in many others. Tcl and Tk are being used by hundreds of companies, large and small, as well as universities and research laboratories.

One benefit that came as a surprise to me is that it is possible to create interesting graphical user interfaces (GUIs) entirely as Tcl scripts. I had always assumed that every Tcl application would contain some new C code that implements new Tcl commands, plus some Tcl scripts that combine the new commands with the built-in facilities provided by Tcl. However, once a simple Tcl/Tk application called `wish` became available, many people began creating user interfaces by writing Tcl scripts for it, without writing any C code at all! It turned out that the Tcl and Tk commands provide a high-level interface to GUI programming that hides many of the details faced by a C programmer. As a result, it is much easier to learn how to use `wish` than a C-based toolkit, and user interfaces can be written with much less code. Most Tcl/Tk users never write any C code at all and most of the Tcl/Tk applications consist solely of Tcl scripts.

This book is intended as an introduction to Tcl and Tk for programmers who plan to write or modify Tcl/Tk applications. I assume that readers have programmed in C and have at least passing familiarity with a shell such as `sh` or `csh` or `ksh`. I also assume that readers have used the X Window System and are familiar with basic ideas such as using the mouse, resizing windows, etc. No prior experience with Tcl or Tk is needed in order to read this book, and you need not have written X applications using other toolkits such as Motif.

The book is organized so that you can learn Tcl without learning Tk if you wish. Also, the discussion of how to write Tcl scripts is separate from the discussion of how to use the C library interfaces provided by Tcl and Tk. The first two parts of the book describe Tcl and Tk at the level of writing scripts, and the last part describes the C interfaces for Tcl; if you are like the majority of Tcl/Tk users who only write scripts, you can stop after reading the first two parts.

In spite of my best efforts, I'm sure that there are errors in this edition of the book. I'm interested in hearing about any problems that you encounter, whether they are typos, formatting errors, sections or ideas that are hard to understand, or bugs in the examples. I'll attempt to correct the problems in future printings of the book.

Many people have helped in the creation of this book. First and foremost I would like to thank Brian Kernighan, who reviewed several drafts of the manuscript with almost terrifying thoroughness and uncovered numerous problems both large and small. I am also grateful for the detailed comments provided by the other Addison-Wesley technical reviewers: Richard Blevins, Gerard Holzmann, Curt Horkey, Ron Hutchins, Stephen Johnson, Oliver Jones, David Korn, Bill Leggett, Don Libes, Kent Margraf, Stuart McRobert, David Richardson, Alexei Rodrigues, Gerald Rosenberg, John Slater, and Win Treese. Thanks also to Bob Sproull, who read the next-to-last draft from cover to cover and provided countless bug fixes and suggestions.

I made early drafts of the manuscript available to the Tcl/Tk community via the Internet and received countless comments and suggestions from all over the world in return. I'm afraid that I didn't keep careful enough records to acknowledge all the people who contributed in this way, but the list of contributors includes at least the following people: Marvin Aguero, Miriam Amos Nihart, Jim Anderson, Frederik Anheuser, Jeff Blaine, John Boller, David Boyce, Terry Brannon, Richard Campbell, J. Cazander, Wen Chen, Richard Cheung, Peter Chubb, De Clarke, Peter Collinson, Peter Costantinidis, Alistair Crooks, Peter Davies, Tal Dayan, Akim Demaille, Mark Diekhans, Matthew Dillon, Tuan Doan, Tony Duarte, Paul DuBois, Anton Eliens, Marc R. Ewing, Luis Fernandes, Martin Forssen, Ben Fried, Matteo Frigo, Andrej Gabara, Steve Gaede, Sanjay Ghemawat, Bob Gibson, Michael Halle, Jun Hamano, Stephen Hansen, Brian Harrison, Marti Hearst, Fergus Henderson, Kevin Hendrix, David Herron, Patrick Hertel, Carsten Heyl, Leszek Holenderski, Jamie Honan, Rob W.W. Hooft, Nick Hounsome, Christopher Hylands, Jonathan Jowett, Poul-Henning Kamp, Karen L. Karavanic, Sunil Khatri, Vivek Khera, Jon Knight, Roger Knopf, Ramkumar Krishnan, Dave Kristol, Peter LaBelle, Tor-Erik Larsen, Tom Legrady, Will E. Leland, Kim Lester, Joshua Levy, Don Libes, Oscar Linares, David C.P. Linden, Toumas J. Lukka, Steve Lord, Steve Lumetta, Earlin Lutz, David J. Mackenzie, B.G. Mahesh, John Maline, Graham Mark, Stuart

McRobert, George Moon, Michael Morris, Russell Nelson, Dale K. Newby, Richard Newton, Peter Nguyen, David Nichols, Marty Olevitch, Rita Ousterhout, John Pierce, Stephen Pietrowicz, Anna Pluzhnikov, Nico Poppelier, M.V.S. Ramanath, Cary D. Renzema, Mark Roseman, Samir Tiongson Saxena, Jay Schmidgall, Dan M. Serachitopol, Hume Smith, Frank Stajano, Larry Streepy, John E. Stump, Michael Sullivan, Holger Teutsch, Bennett E. Todd, Glenn Trewitt, D.A. Vaughan-Pope, Richard Vieregge, Larry W. Virden, David Waitzman, Matt Wartell, Glenn Waters, Wally Wedel, Juergen Weigert, Mark Weiser, Brent Welch, Alex Woo, Su-Lin Wu, Kawata Yasuro, Chut Ngeow Yee, Richard Yen, Stephen Ching-SingYen, and Mike Young.

Many many people have made significant contributions to the development of Tcl and Tk. Without all of their efforts there would have been nothing of interest to write about in this book. Although I cannot hope to acknowledge all the people who helped to make Tcl and Tk what they are today, I would like to thank the following people specially: Don Libes, for writing the first widely used Tcl application; Mark Diekhans and Karl Lehenbauer, for `TclX`; Alastair Fyfe, for supporting the early development of Tcl; Mary Ann May-Pumphrey, for developing the original Tcl test suite; George Howlett, Michael McLennan, and Sani Nassif, for the `BLT` extensions; Kevin Kenny, for showing that Tcl can be used to communicate with almost any imaginable program; Joel Bartlett, for many challenging conversations and for inspiring Tk's canvas widget with his `ezd` program; Larry Rowe, for developing `Tcl-DP` and for providing general advice and support; Sven Delmas, for developing the `XF` application builder based on Tk; and Andrew Payne, for the widget tour and for meritorious Tcl evangelism.

Several companies have provided financial support for the development of Tcl and Tk, including Digital Equipment Corporation, Hewlett-Packard Corporation, Sun Microsystems, and Computerized Processes Unlimited. I am particularly grateful to Digital's Western Research Laboratory and its director, Richard Swan, for providing me with a one-day-per-week hideaway where I could go to gather my thoughts and work on Tcl and Tk.

Terry Lessard-Smith and Bob Miller have provided fabulous administrative support for this and all my other projects. I don't know how I would get anything done without them.

Finally, I owe a special debt to my colleague and friend Dave Patterson, whose humor and sage advice have inspired and shaped much of my professional career, and to my wife, Rita, and daughters, Kay and Amy, who have tolerated my workaholic tendencies with more cheer and affection than I deserve.

John Ousterhout
Berkeley, California
February 1994

Introduction

This book is about two software packages called Tcl and Tk.[1] Tcl is a *dynamic language* (also known as a *scripting language*) for controlling and extending applications; its name stands for “tool command language.” Tcl provides general programming facilities sufficient for most applications. Furthermore, Tcl is both *embeddable* and *extensible*. Its interpreter is a library of C functions that can easily be incorporated into applications, and each application can extend the core Tcl features with additional commands either unique to the application or provided by add-on libraries (referred to as *extensions* in the Tcl community).

One of the most useful extensions to Tcl is Tk, which is a toolkit for developing graphical user interface (GUI) applications. Tk extends the core Tcl facilities with commands for building user interfaces, so that you can construct GUIs by writing Tcl scripts instead of C code. Like Tcl, Tk is implemented as a library of C functions so it can be used in many different applications.

Note: This book corresponds to Tcl/Tk version 8.5. The release notes for each version of Tcl/Tk describe the changes and new features in each release. The Tcler’s Wiki web site (`http://wiki.tcl.tk`) also compiles change lists for each release; you can find them by searching for pages that contain *Changes in* in the title.

1. The official pronunciation for Tcl is “tickle,” although “tee-see-ell” is also used frequently. Tk is pronounced “tee-kay.”

I.1 Benefits of Tcl/Tk

Together, Tcl and Tk provide several benefits to application developers and users. The first benefit is rapid development. Many interesting applications can be written entirely as Tcl scripts. This allows you to program at a much higher level than you would in C/C++ or Java, and Tk hides many of the details that C or Java programmers must address. Compared to low-level toolkits, there is much less to learn in order to use Tcl and Tk, and much less code to write. New Tcl/Tk users often can create interesting user interfaces after just a few hours of learning, and many people have reported tenfold reductions in code size and development time when they switched from other toolkits to Tcl and Tk.

Another reason for rapid development with Tcl and Tk is that Tcl is an interpreted language. When you use a Tcl application, you can generate and execute new scripts on the fly without recompiling or restarting the application. This allows you to test out new ideas and fix bugs rapidly. Since Tcl is interpreted, it executes more slowly than compiled C code; but internal optimizations, such as bytecode compilation coupled with ever-increasing processor power, have erased most of the perceived performance advantages of compiled languages. For example, you can execute scripts with hundreds of Tcl commands on each movement of the mouse with no perceptible delay. In the rare cases where performance becomes an issue, you can reimplement the performance-critical parts of your Tcl scripts in C.

A second benefit is that Tcl is a cross-platform language, as are most of its extensions, including Tk. This means that an application developed on one platform, such as Linux, in most cases can be run without change on another platform, such as Macintosh or Windows.

Tcl was also the first dynamic language to have native Unicode support. As a result, Tcl applications can handle text in virtually any of the world's written languages. Tcl requires no extensions to process text in any of the Unicode-supported scripts, and standard extensions such as `msgcat` provide simple localization support.

Another significant benefit is that Tcl and most of its extensions are freely available as open source. Tcl and Tk follow the so-called BSD license, which allows anyone to download, inspect, modify, and redistribute Tcl/Tk without charge.

Tcl is an excellent "glue language." A Tcl application can include many different extensions, each of which provides an interesting set of Tcl commands. Tk is one example of a library package; many other packages have been developed by the Tcl/Tk community, and you can also write your own packages. Tcl scripts for such applications can include commands from any of the packages.

Additionally, Tcl makes it easy for applications to have powerful scripting languages. For example, to add scripting capability to an existing application, all you need do is implement a few new Tcl commands that provide the basic features of the application. Then you can link your new commands with the Tcl library to produce a full-function scripting language that includes both the commands provided by Tcl (called the *Tcl core*) and those that you wrote.

Tcl also provides user convenience. Once you learn Tcl and Tk, you will be able to write scripts for any Tcl and Tk application merely by learning the few application-specific commands for the new application. This should make it possible for more users to personalize and enhance their applications.

I.2 Organization of the Book

Chapter 1 uses several simple scripts to provide a quick overview of the most important features of Tcl and Tk. It is intended to give you the flavor of the systems and convince you that they are useful, without explaining anything in detail. The remainder of the book goes through everything again in a more comprehensive fashion. It is divided into three parts:

- **Part I** introduces the Tcl scripting language. After reading this section, you will be able to write scripts for Tcl applications. You will need to know at least some of the information in this part in order to write Tk applications.
- **Part II** describes the additional Tcl commands provided by Tk, which allow you to create user-interface widgets such as menus and scrollbars and arrange them in GUI applications. After reading this section, you will be able to create new GUI applications and write scripts to enhance existing Tk applications.
- **Part III** discusses the C functions in the Tcl library and how to use them to create new Tcl commands. After reading this section, you will be able to write new Tcl packages and applications in C. However, you will be able to do a great deal (perhaps everything you need) without this information.

Each of these parts contains about a dozen short chapters. Each chapter is intended to be a self-contained description of a piece of the system, and you need not necessarily read the chapters in order.

Not every feature of Tcl and Tk is covered here, and the explanations are organized to provide a smooth introduction rather than a complete reference source. A separate set of reference manual entries, referred to as the *reference documentation*, is available with the Tcl and Tk distributions. These are much more terse but they cover absolutely every feature of both systems. Appendix A describes how to retrieve the Tcl and Tk distributions, including reference documentation, via the Internet. Appendix B provides a survey of some popular Tcl extensions. Appendix C lists additional online and printed resources for Tcl and Tk. The full Tcl Source Distribution License is included in Appendix D.

This book assumes that you already know how to use your operating system, including interacting with applications from the command line. Part III of this book assumes that you are familiar with the C programming language as defined by the ANSI C standard; a basic knowledge of C is helpful for Parts I and II but not required. You need not know anything about either Tcl or Tk before reading this book; both are introduced from scratch.

I.3 Notation

This book uses a `monospace` font for anything that might be typed to a computer, such as Tcl scripts, C code, and names of variables, procedures, and commands. The examples of Tcl scripts use notation like the following:

```
   set a 44
⇒ 44
```

Tcl commands, such as `set a 44` in the example, appear in `monospace`; their results, such as *`44`* in the example, appear in italicized *`monospace`*. The ⇒ symbol before the result indicates that this is a normal return value. If an error occurs in a Tcl command, the error message appears in italicized *`monospace`*, preceded by a ∅ symbol to indicate that this is an error rather than a normal return:

```
   set a 44 55
∅ wrong # args: should be "set varName ?newValue?"
```

When describing the syntax of Tcl commands, italicized Courier is used for formal argument names. An argument or group of arguments enclosed in question marks indicates that the arguments are optional. For example, the syntax of the `set` command is as follows:

```
set varName ?newValue?
```

This means that the word `set` must be entered verbatim to invoke the command, and *`varName`* and *`newValue`* are the names of `set`'s arguments; when invoking the command, you type a variable name instead of *`varName`* and a new value for the variable instead of *`newValue`*. The *`newValue`* argument is optional.

1

An Overview of Tcl and Tk

This chapter introduces Tcl and Tk with a series of scripts illustrating their main features. Although you should be able to start writing simple scripts after reading this chapter, the explanations here are not complete. The purpose of this chapter is to show you the overall structure of Tcl and Tk and the kinds of things they can do, so that when individual features are discussed in detail you'll be able to see why they are useful. All of the information in this chapter is revisited in more detail in later chapters, and several important aspects, such as the Tcl C interfaces, are not discussed at all in this chapter.

1.1 Getting Started

To invoke Tcl scripts, you must run a Tcl application. If Tcl is installed on your system, there should exist a simple Tcl shell application called `tclsh`, which you can use to try out some of the examples in this chapter. If Tcl has not been installed on your system, refer to Appendix A for information on how to obtain and install it.

Note: It's common practice to install `tclsh` with its version number as part of the name (for example, `tclsh8.5` on Unix or `tclsh85` on Windows). 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 uniformly across different versions of Tcl. Therefore, most installers also commonly link or alias `tclsh` to the most recent version installed on the system. The same is true for the `wish` interpreter, described later. Therefore, unless you want to use a specific version of a Tcl application installed on your system, you should simply use `tclsh` or `wish`.

You can start the `tclsh` application by opening a terminal window on a Macintosh or Unix system, or a command prompt window on a Windows system, and then entering the command

```
tclsh
```

This causes `tclsh` to start in interactive mode, reading Tcl commands from the keyboard and passing them to the Tcl interpreter for evaluation. For starters, enter the following command at the `tclsh` prompt:

```
expr 2 + 2
```

`tclsh` prints the result (`4`) and then prompts you for another command.

This example illustrates several features of Tcl. Each command consists of one or more *words* separated by spaces or tabs (referred to as *whitespace* characters). In the example there are four words: `expr`, `2`, `+`, and `2`. The first word of each command is the name of the command to execute. The other words are *arguments* that are passed to the command for processing. `expr` is one of the core commands provided by the Tcl library, so it exists in every Tcl application. It concatenates its arguments into a single string and evaluates the string as an arithmetic expression.

Each Tcl command returns a result. If a command has no meaningful result, it returns an empty string. For the `expr` command the result is the value of the expression.

All values in Tcl have a string representation and may also have a more efficient internal representation. In this example, `expr`'s result is a numerical value that would have a binary integer or floating-point internal representation. The internal representation allows faster and more efficient processing of information. If the value simply is assigned to a variable or if it is used by another command expecting a numerical value, this is done very efficiently as no string conversion is required. Tcl automatically generates a string representation of a value on an as-needed basis—for example, when the value is displayed on the console.

Note: At the script development level, you can treat all values as strings; Tcl converts between the string and the internal representation automatically as needed. As you grow more familiar with Tcl, understanding what can cause conversions can help you avoid them, resulting in more efficient and faster code. In general, always being consistent in your treatment of a value (e.g., always using list commands to process lists, always using dictionary commands to process dictionaries, etc.) and avoiding unnecessary printing and other string manipulation of numeric, list, and dictionary values can go a long way in speeding up your code. For more information on the internal representation of values at the C language level, see Chapter 32.

From now on, we will use notation such as the following to describe examples:

```
  expr 2 + 2
⇒ 4
```

The first line is the command you enter and the second line is the result returned by the command. The ⇒ symbol indicates that the line contains a return value; the ⇒ is not actually printed out by `tclsh`. We will omit return values in cases where they aren't important, such as sequences of commands where only the last command's result matters.

Commands are normally terminated by newlines (typically the Enter or Return key on your keyboard), so each line that you enter in `tclsh` normally becomes a separate command. Semicolons also act as command separators, in case you wish to enter multiple commands on a single line. It is also possible for a single command to span multiple lines; you'll see how to do this later.

The `expr` command supports an expression syntax similar to that of expressions in ANSI C, including the same precedence rules and most of the C operators. Here are a few examples that you could enter in `tclsh`:

```
  expr 2 * 10 - 1
⇒ 19
  expr 14.1*6
⇒ 84.6
  expr sin(.2)
⇒ 0.19866933079506122
  expr rand()
⇒ 0.62130973004797
  expr rand()
⇒ 0.35263291623100307
  expr (3 > 4) || (6 <= 7)
⇒ 1
```

The first example shows the multiplication operator and how it has a higher precedence than subtraction. The second shows that expressions can contain real values as well as integer values. The next examples show some of the built-in functions supported by `expr`, including the `rand()` function for generating random numbers between 0 and 1. The last example shows the use of the relational operators > and

`<=` and the logical OR operator `||`. As in C, Boolean results are represented numerically with `1` for true and `0` for false.

To leave `tclsh`, invoke the `exit` command:

```
exit
```

This command terminates the application and returns you to your shell.

1.2 "Hello, World!" with Tk

Tcl provides a full set of programming features such as variables, loops, and procedures. It can be used by itself or with extensions that implement their own Tcl commands in addition to those in the Tcl core.

One of the more interesting extensions to Tcl is the set of windowing commands provided by the Tk toolkit. Tk's commands allow you to create graphical user interfaces. Many of the examples in this book use an application called `wish` ("windowing shell"), which is similar to `tclsh` except that it also includes the commands defined by Tk. If Tcl and Tk have been installed on your system, you can invoke `wish` from your terminal or command prompt window just as you did for `tclsh`; it displays a small empty window on your screen and then reads commands from the console. Alternatively, if you have Tcl/Tk version 8.4 or later installed, you can invoke the `tclsh` application, and then use the command `package require Tk` to dynamically load the Tk extension.

Note: On Windows, invoking an interactive `wish` session displays both the empty window and a separate console window. The console window is a replacement for a real console to allow input and output on the standard I/O channels. The console window normally is hidden when a script file is executing, as described later, although you can display it by executing the `console show` command. Consult the `console` reference documentation for more information.

Here is a simple Tk script that you could run with `wish`:

```
button .b -text "Hello, world!" -command exit
grid .b
```

If you enter these two Tcl commands in `wish`, the window's appearance changes to that shown in Figure 1.1. If you move the pointer over the "Hello, world!" text and click the main mouse button (the leftmost button in most configurations), the window disappears and `wish` exits.

Figure 1.1 The "Hello, world!" application

Several things about this example need explanation. First let us deal with the syntactic issues. The example contains two commands, `button` and `grid`, both of which are implemented by Tk. Although these commands do not look like the `expr` command in the previous section, they have the same basic structure as all Tcl commands: one or more words separated by whitespace characters. The `button` command contains six words, and the `grid` command contains two words.

The fourth word of the `button` command is enclosed in double quotes. This allows the word to include whitespace characters; without the quotes, `Hello,` and `world!` would be separate words. The double quotes are delimiters, not part of the word itself; they are removed by the Tcl interpreter before the command is executed.

For the `expr` command the word structure doesn't matter much since `expr` concatenates all its arguments. However, for the `button` and `grid` commands, and for most Tcl commands, the word structure is important. The `button` command expects its first argument to be the name of a new window to create. Additional arguments to this command must come in pairs, where the first argument of each pair is the name of a *configuration option* and the second argument is a value for that option. Thus if the double quotes were omitted, the value of the `-text` option would be `Hello,` and `world!` would be treated as the name of a separate configuration option. Since there is no option defined with the name `world!` the command would return an error.

Now let us move on to the behavior of the commands. The basic building block for a graphical user interface in Tk is a *widget*. A widget is a window with a particular appearance and behavior (the terms *widget* and *window* are used synonymously in Tk). Widgets are divided into classes such as buttons, menus, and scrollbars. All the widgets in the same class have the same general appearance and behavior. For example, all button widgets display a text string, bitmap, or image and execute a Tcl script when the user clicks the button.

Widgets are organized hierarchically in Tk, with names that reflect their positions in the hierarchy. The *main widget*, which appeared on the screen when you started `wish`, has the name `.` and `.b` refers to a child `b` of the main widget. Widget names in Tk are like file name paths except that they use `.` as a separator character instead of / or \. Thus, `.a.b.c` refers to a widget that is a child of widget `.a.b`, which in turn is a child of `.a`, which is a child of the main widget.

Tk provides one command for each class of widgets, called a *class command*, which you invoke to create widgets of that class. For example, the `button` command creates button widgets. This is similar to standard object-oriented programming principles, though Tk doesn't support direct subclassing of the widget classes. All of the class commands have the same form: the first argument is the name of a new widget to create, and additional arguments specify configuration options. Different widget classes support different sets of options. Widgets typically have many options, with default values for the options that you don't specify. When a class command like `button` is invoked, it creates a new widget with the given name and configures it as specified by the options.

The `button` command in the example specifies two options: `-text`, which is a string to display in the button, and `-command`, which is a Tcl script to execute when the user invokes the button. In this example the `-command` option is `exit`. Here are a few other button options that you can experiment with:

- `-background`—the background color for the button, such as `blue`
- `-foreground`—the color of the text in the button, such as `black`
- `-font`—the font to use for the button, such as `"times 12"` for a 12-point Times Roman font

Creating a widget does not automatically cause it to be displayed. The `grid` command causes the button widget to appear on the screen. Independent entities called *geometry managers* are responsible for computing the sizes and locations of widgets and making them appear on the screen. The separation of widget creation and geometry management provides significant flexibility in arranging widgets on the screen to design your application. The `grid` command in the example asks a geometry manager called the *gridder* to manage `.b`. The gridder arranges widgets in a grid of columns and rows. In this case, the command placed `.b` in the first column of the first row of the grid and sized the grid to just large enough to accommodate the widget; furthermore, if the parent has more space than needed by the grid, as in the example, the parent is shrunk so that it is just large enough to hold the child. Thus, when you entered the `grid` command, the main window (`.`) shrank from its original size to the size that appears in Figure 1.1.

1.3 Script Files

In the examples so far, you have entered Tcl commands interactively to `tclsh` or `wish`. You can also place commands into script files and invoke the script files just like shell scripts. To do this for the "Hello, world!" example, place the following text in a file named `hello.tcl`:

```
#!/usr/local/bin/wish
button .b -text "Hello, world!" -command exit
pack .b
```

You can execute this script by invoking the wish interpreter and passing the script file name as a command-line argument:

```
wish hello.tcl
```

This causes wish to display the same window as shown in Figure 1.1 and wait for you to interact with it. In this case you will not be able to type commands interactively to wish; all you can do is click on the button.

1.3.1 Executable Scripts on Unix and Mac OS X

The script just shown is the same as the one you typed earlier except for the first line. As far as wish is concerned, this line is a comment, but on Unix systems if you make the file executable (for example, by executing `chmod +x hello.tcl` in your shell), you can then invoke the file directly by typing `hello.tcl` to your shell. (This requires the directory containing your `hello.tcl` script to be listed in your `PATH` environment variable.) When you do this, the system invokes wish, passing it the file as a script to interpret.

As written, this script works as an executable script only if wish is installed in `/usr/local/bin`, although you could still run it by invoking wish with the script file name as a command-line argument. If wish has been installed somewhere else, you need to change the first line to reflect its location on your system. Some systems misbehave in confusing ways if the first line of the script file is longer than 32 characters, so beware if the full path name of the wish binary is longer than 27 characters.

To work around these limitations, a common technique for scripts on Unix has been to start script files with the following three lines:

```
#!/bin/sh
# Tcl ignores the next line but 'sh' doesn't \
exec wish "$0" "$@"
```

or the more arcane but more robust version:

```
#!/bin/sh
# Tcl ignores the next line but 'sh' doesn't \
exec wish "$0" ${1+"$@"}
```

In most modern Unix implementations, though, the following will work correctly, as long as wish appears in one of the directories in your `PATH` environment variable:

```
#!/usr/bin/env wish
```

1.3.2 Executable Scripts on Windows

On Windows, you can use the standard system tools to associate the wish interpreter with a file extension (`.tcl` by convention) so that double-clicking on the icon for a Tcl/Tk script automatically invokes the `wish` interpreter, passing it the name of the file as a script to interpret. Most Windows installers for Tcl/Tk automatically create this association for you. `wish` is typically selected as the default association because most Windows-based Tcl/Tk programs are GUI-based. However, if the majority of your Tcl scripts don't use Tk commands, you could change the default association to invoke `tclsh`.

If you plan to distribute your scripts on multiple platforms, you should include the appropriate `#!` header as discussed in the previous section for Unix executable scripts so that they can be directly executable on Unix systems. On the other hand, Windows doesn't follow the `#!` convention, and the `#!` line is treated as a comment by the `wish` interpreter, so the net effect is that the line is ignored when the script is run on a Windows system.

1.3.3 Executing Scripts in an Interactive Interpreter

In practice, users of Tk applications rarely type Tcl commands; they interact with the applications using the mouse and keyboard in the usual ways you would expect for graphical applications. Tcl works behind the scenes where users don't normally see it. The `hello.tcl` script behaves just the same as an application that has been coded in C with a GUI toolkit and compiled into a binary executable file.

During debugging, though, it is common for application developers to type Tcl commands interactively. For example, you could test the `hello.tcl` script by starting `wish` interactively (type `wish` to your shell instead of `hello.tcl`). Then enter the following Tcl command:

```
source hello.tcl
```

`source` is a Tcl command that takes a file name as an argument. It reads the file and evaluates it as a Tcl script. This generates the same user interface as if you had invoked `hello.tcl` directly from your shell, but you can now enter Tcl commands interactively, too. For example, you could edit the script file to change the `-command` option to

```
-command "puts Good-bye!; exit"
```

then enter the following commands interactively to `wish` without restarting the program:

```
destroy .b
source hello.tcl
```

The first command deletes the existing button, and the second command re-creates the button with the new `-command` option. Now when you click on the button, the `puts` command prints a message on standard output before `wish` exits.

1.4 Variables and Substitutions

Tcl allows you to store values in variables and use those values in commands. For example, consider the following script, which you could enter in either `tclsh` or `wish`:

```
  set a 44
⇒ 44
  expr $a*4
⇒ 176
```

The first command assigns the value `44` to the variable `a` and returns the variable's value. In the second command, the `$` causes Tcl to perform *variable substitution*: the Tcl interpreter replaces the dollar sign and the variable name following it with the value of the variable, so that the actual argument received by `expr` is `44*4`. Variables need not be declared in Tcl; they are created automatically when set. Variable values can always be represented as strings but may be maintained in a native binary format. Strings may contain binary data and may be of any length. Of course, in this example an error occurs in `expr` if the value of `a` doesn't make sense as an integer or real number.

Tcl also provides *command substitution*, which allows you to use the result of one command in an argument to another command:

```
  set a 44
  set b [expr $a*4]
⇒ 176
```

Square brackets invoke command substitution: everything inside the brackets is evaluated as a separate Tcl script, and the result of that script is substituted into the word in place of the bracketed command. In this example the second argument of the second `set` command is `176`.

The final form of substitution in Tcl is *backslash substitution*, which either adds special meaning to a normal character or takes it away from a special character, as in the following examples:

```
  set x \$a
  set newline \n
```

The first command sets the variable `x` to the string `$a` (the characters `\$` are replaced with a dollar sign and no variable substitution occurs). The second command sets the variable `newline` to hold a string consisting of the newline character (the characters `\n` are replaced with a newline character).

1.5 Control Structures

The next example uses variables and substitutions along with some simple control structures to create a Tcl *procedure* called `factorial`, which computes the factorial of a given non-negative integer value:

```
proc factorial {val} {
    set result 1
    while {$val>0} {
        set result [expr $result*$val]
        incr val -1
    }
    return $result
}
```

If you enter the preceding lines in `wish` or `tclsh`, or if you enter them into a file and then `source` the file, a new command `factorial` becomes available. The command takes one non-negative integer argument, and its result is the factorial of that number:

```
  factorial 3
⇒ 6
  factorial 20
⇒ 2432902008176640000
  factorial 0.5
∅ expected integer but got "0.5"
```

This example uses one additional piece of Tcl syntax: braces. Braces are like double quotes in that they can be placed around a word that contains embedded spaces. However, braces are different from double quotes in two respects. First, braces nest. The last word of the `proc` command starts after the open brace on the first line and contains everything up to the close brace on the last line. The Tcl interpreter removes the outer braces and passes everything between them, including several nested pairs of braces, to `proc` as an argument. The second difference between braces and double quotes is that no substitutions occur inside braces, whereas they do inside quotes. All of the characters between the braces are passed verbatim to `proc` without any special processing.

The `proc` command takes three arguments: the name of a procedure, a list of argument names separated by whitespace, and the body of the procedure, which is a Tcl script. `proc` enters the procedure name into the Tcl interpreter as a new command. Whenever the command is invoked, the body of the procedure is evaluated. While the procedure body is executing, it can access its arguments as variables: `val` holds the first and only argument.

The body of the `factorial` procedure contains three Tcl commands: `set`, `while`, and `return`. The `while` command does most of the work of the procedure.

It takes two arguments, an expression, `$val>0`, and a body, which is another Tcl script. The `while` command evaluates its expression argument and if the result is nonzero, it evaluates the body as a Tcl script. It repeats this process over and over until eventually the expression evaluates to zero. In the example, the body of the `while` command multiplies the result by `val` and then uses the `incr` command to add the specified integer increment (`-1` in this case) to the value contained in `val`. When `val` reaches zero, `result` contains the desired factorial.

The `return` command causes the procedure to exit with the value of the variable `result` as the procedure's result. If a `return` command is omitted, the return value of a procedure is the result of the last command executed in the procedure's body. In the case of `factorial` this would be the result of `while`, which is always an empty string.

The use of braces in this example is crucial. The single most difficult issue in writing Tcl scripts is managing substitutions: making them happen when you want them and preventing them when you don't. The body of the procedure must be enclosed in braces because we don't want variable and command substitutions to occur at the time the body is passed to `proc` as an argument; we want the substitutions to occur later, when the body is evaluated as a Tcl script. The body of the `while` command is enclosed in braces for the same reason: rather than performing the substitutions once, while parsing the `while` command, we want the substitutions to be performed over and over, each time the body is evaluated. Braces are also needed in the `{$val>0}` argument to `while`. Without them the value of the variable `val` would be substituted when the `while` command is parsed; the expression would have a constant value and `while` would loop forever. Try replacing some of the braces in the example with double quotes to see what happens.

The examples in this book use a style in which the open brace for an argument that is a Tcl script appears at the end of one line, the script follows on successive indented lines, and the close brace is on a line by itself after the script. Although this makes for readable scripts, Tcl doesn't require this particular syntax. Arguments that are scripts are subject to the same syntax rules as any other arguments; in fact, the Tcl interpreter doesn't even know that an argument is a script at the time it parses it. One consequence is that the open brace must be on the same line as the preceding portion of the command. If the open brace is moved to a line by itself, the newline before the open brace terminates the command.

The variables in a procedure are normally local to that procedure and are not visible outside the procedure. In the `factorial` example the local variables include the argument `val` as well as the variable `result`. A fresh set of local variables is created for each call to a procedure (arguments are passed by copying their values), and when a procedure returns, its local variables are deleted. Variables named outside any procedure are called *global variables*; they last forever unless explicitly deleted. You'll find out later how a procedure can access global variables and the local variables of other active procedures. Additionally, persistent variables can be created within specific *namespaces* to prevent naming conflicts; Chapter 10 discusses the use of namespaces.

1.6 On the Tcl Language

As a programming language, Tcl is defined quite differently from most other languages. Most languages have a grammar that defines the entire language. For example, consider the following statement in C:

```
while (val>0) {
    result *= val;
    val -= 1;
}
```

The grammar for C defines the structure of this statement in terms of a reserved word `while`, an expression, and a substatement to execute repeatedly until the expression evaluates to zero. The C grammar defines both the overall structure of the `while` statement and the internal structure of its expression and substatement.

In Tcl no fixed grammar explains the entire language. Instead, Tcl is defined by an interpreter that parses single Tcl commands, plus a collection of procedures that execute individual commands. The interpreter and its substitution rules are fixed, but new commands can be defined at any time and existing commands can be replaced. Features such as control flow, procedures, and expressions are implemented as commands; they are not understood directly by the Tcl interpreter. For example, consider the Tcl command that is equivalent to the preceding `while` loop:

```
while {$val>0} {
    set result [expr $result*$val]
    incr val -1
}
```

When this command is evaluated, the Tcl interpreter knows nothing about the command except that it has three words, the first of which is a command name. The Tcl interpreter has no idea that the first argument to `while` is an expression and the second is a Tcl script. Once the command has been parsed, the Tcl interpreter passes the words of the command to `while`, which treats its first argument as an expression and the second as a Tcl script. If the expression evaluates to nonzero, `while` passes its second argument back to the Tcl interpreter for evaluation. At this point the interpreter treats the contents of the argument as a script (i.e., it performs command and variable substitutions and invokes the `expr`, `set`, and `incr` commands).

Now consider the following command:

```
set {$val>0} {
    set result [expr $result*$val]
    incr val -1
}
```

As far as the Tcl interpreter is concerned, the `set` command is identical to the `while` command except that it has a different command name. The interpreter handles this command in exactly the same way as the `while` command, except that it invokes a different procedure to execute the command. The `set` command treats its first argument as a variable name and its second argument as a new value for that variable, so it will set a variable with the rather unusual name of `$val>0`.

The most common mistake made by new Tcl users is to try to understand Tcl scripts in terms of a grammar; this leads people to expect much more sophisticated behavior from the interpreter than actually exists. For example, a C programmer using Tcl for the first time might think that the first pair of braces in the `while` command serves a different purpose from the second pair. In reality, there is no difference. In each case the braces are present so that the Tcl interpreter passes the characters between the braces to the command without performing any substitutions.

Thus the entire Tcl "language" consists of about a dozen simple rules for parsing arguments and performing substitutions. The actual behavior of a Tcl script is determined by the commands executed. The commands determine whether to treat an argument as a literal value, the name of a variable, a code block to execute, and so on. An interesting consequence of this is that a script can define commands implementing entirely new control structures, which is a feature not available in most other languages.

1.7 Event Bindings

The next example provides a graphical front end for the `factorial` procedure. In addition to demonstrating two new widget classes, it illustrates Tk's *binding* mechanism. A binding causes a particular Tcl script to be evaluated whenever a particular event occurs in a particular window. The `-command` option for buttons is an example of a simple binding implemented by a particular widget class. Tk also includes a more general mechanism that can be used to extend the behavior of widgets in nearly arbitrary ways.

To run the example, copy the following script into a file `factorial.tcl` and invoke the file from your shell.

```
#!/usr/bin/env wish
proc factorial {val} {
    set result 1
    while {$val>0} {
        set result [expr $result*$val]
        incr val -1
    }
    return $result
}
entry .value -width 6 -relief sunken -textvariable value
label .description -text "factorial is"
label .result -textvariable result
button .calculate -text "Calculate" \
    -command {set result [factorial $value]}
bind .value <Return> {
    .calculate flash
    .calculate invoke
}
grid .value .description .result -padx 1m -pady 1m
grid .calculate - - -padx 1m -pady 1m
```

This script produces a screen display like that in Figure 1.2. There is an entry widget in which you can click with the mouse and type a number. If you click the button labeled "Calculate," the result appears on the right side of the window; the same occurs if you press the Return key in the entry.

This application consists of four widgets: one entry, one button, and two labels. Entries are widgets that display one-line text strings that you can edit interactively. The entry is configured with a `-width` of `6`, which means it is large enough to display about six digits, and a `-relief` of `sunken`, which makes the entry appear sunken into the window. The `-textvariable` option for each entry specifies the name of a global variable to hold the entry's text—any changes you make in the entry are reflected in the variable and vice versa.

The `.description` label widget holds decorative text, and the `.result` label holds the result of the power computation. The `-textvariable` option for `.result` causes it to display whatever string is in the global variable `result` and to update itself whenever the variable changes. In contrast, `.description` displays a constant string.

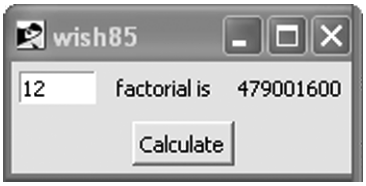

Figure 1.2 A graphical user interface that computes a factorial

The first `grid` command arranges the entry and two label widgets in a row from left to right. The `-padx` and `-pady` options make the display a bit more attractive by arranging for 1 millimeter of extra space on the left and right sides of each widget, and 1 millimeter of extra space above and below each widget. The `m` suffix specifies millimeters; you could also use `c` for centimeters, `i` for inches, `p` for points, or no suffix for pixels.

The second `grid` command arranges the button in a second row. Because the widget name occurs as the first argument, the gridder allocates the first column of the row to the button. The two `-` arguments following the widget name indicate to the gridder that the space allocated to the button widget should span two additional columns. The gridder then centers the button widget inside its allocated space.

The command creating the `.calculate` button occupies two lines in the script; the backslash at the end of the first line is a line-continuation character, which causes the newline to be treated as a space. The button's `-command` script connects the user interface to the `factorial` procedure. The script invokes `factorial`, passing it the values in the entry and storing the result in the `result` variable so that it is displayed in the `.result` widget.

The `bind` command has three arguments: the name of a widget, an event specification, and a Tcl script to invoke when the given event occurs in the given widget. `<Return>` specifies an event consisting of the user pressing the return key on the keyboard (which is still labeled "Return" on Mac keyboards but typically labeled "Enter" on most other English keyboards these days). Table 1.1 shows a few other event specifiers that you might find useful.

Table 1.1 Event Specifiers

Event specifer	Meaning
`<Button-1>`	Mouse button 1 is pressed
`<1>`	Shorthand for `<Button-1>`
`<ButtonRelease-1>`	Mouse button 1 is released
`<Double-Button-1>`	Double-click on mouse button 1
`<Key-a>`	Key `a` is pressed
`<a>` or `a`	Shorthand for `<Key-a>`
`<Motion>`	Pointer motion with any (or no) buttons or modifier keys pressed
`<B1-Motion>`	Pointer motion with button 1 pressed

The script for a binding has access to several pieces of information about the event, such as the location of the pointer when the event occurred. For an example, start up `wish` interactively and enter the following command in it:

```
bind . <Motion> {puts "pointer at %x,%y"}
```

Now move the pointer over the window. Each time the pointer moves, a message is printed on standard output giving its new location. When the pointer motion event occurs, Tk scans the script for `%` sequences and replaces them with information about the event before passing the script to Tcl for evaluation. `%x` is replaced with the pointer's x-coordinate and `%y` is replaced with the pointer's y-coordinate.

The intent of a binding is to extend the generic built-in behavior of the entry (editing text strings) with an application-specific behavior. In this script, as a convenience we would like to allow the user to request the factorial calculation by pressing the Return key as an alternative to clicking the "Calculate" button. We could simply duplicate the button's command script, but if we were to modify the command script later, we'd need to remember to replicate the change in the binding script as well. Instead, we provide a binding script that "programmatically clicks" the button.

The binding script executes two commands called *widget commands*. Whenever a new widget is created, a new Tcl command is also created with the same name as the widget, and you can invoke this command to communicate with the widget. The first argument to a widget command selects one of several operations, and additional arguments are used as parameters for that operation. In this binding script, the first widget command flashes the button. (Depending on your system's color scheme, you might not see the button flash.) The second widget command causes the button widget to invoke its `-command` option just as if you had clicked the mouse button on it.

Each class of widget supports a different set of operations in its widget commands, but many of the operations are similar from class to class. For example, every widget class supports a `configure` widget command that can be used to modify any of the configuration options for the widget. If you run the `factorial.tcl` script interactively, you could type the following command to change the background of the entry widget to yellow:

```
.value configure -background yellow
```

Or you could type

```
.calculate configure -state disabled
```

to make the button unresponsive to user interaction.

1.8 Additional Features of Tcl and Tk

The examples in this chapter have used almost every aspect of the Tcl language syntax, and they illustrated many features of Tcl and Tk. However, Tcl and Tk contain many other facilities that are not used in this chapter; all of these are described later in the book. Here is a sample of some of the most useful features that haven't been mentioned yet:

- **Arrays, dictionaries, and lists**—Tcl provides associative arrays and dictionaries for storing key-value pairs efficiently and lists for managing aggregates of data.
- **More control structures**—Tcl provides several additional commands for controlling the flow of execution, such as `eval`, `for`, `foreach`, and `switch`.
- **String manipulation**—Tcl contains a number of commands for manipulating strings, such as measuring their length, regular expression pattern matching and substitution, and format conversion.
- **File access**—You can read and write files from Tcl scripts and retrieve directory information and file attributes such as size and creation time.
- **More widgets**—Tk contains many widget classes besides those shown here, such as menus, scrollbars, a drawing widget called a *canvas*, and a text widget that makes it easy to achieve hypertext effects.
- **Access to other windowing features**—Tk provides commands for accessing all of the major windowing facilities, such as a command for communicating with the window manager (to set the window's title, for example), a command for retrieving the selection, and a command to manage the input focus.
- **Interapplication communication**—Tcl includes the ability to communicate between applications through interprocess pipes and TCP/IP sockets.
- **C interfaces**—Tcl provides C library procedures that you can use to define new Tcl commands in C. (Tk provides a library that you can use to create new widget classes and geometry managers in C, but this capability is rarely used and so is not covered in this book.)

Index

Symbols

!! in history shortcuts, 286
# in comments, 32–35
#! in script execution, 9–10
$
  command evaluation, 24
  as constraint, 80
  invoking substitution, 11
  variable substitution, 24–25, 43–45
%
  in expressions, 55–56
  substitutions in scripts that handle events, 419–421
  in time and date strings, 254–256
& in expressions, 56, 57
&& in expressions, 56, 57
*
  expr syntax, 5
  glob-style pattern matching, 79
  operators and precedence, 55–56
  as quantifier, 84
  in regular expressions, 81
** in expressions, 55–56
:: in qualified names, 161–162
  ::pkg::create, 237
::tcl::tm::path list, 231, 240–241
::tk, 304
::ttk, 304
@ in image options, 311
[ ]
  command substitution, 26
  invoking substitution, 11
  regular expression atoms, 82–83
\\ in substitution, 27
^
  as constraint, 80
  in gridder, 396
  operators and precedence, 56, 57
^old^new, 286
{ }
  basic dictionary structure, 118
  in glob commands, 180
  inside comments, 34–35
  in lists, 104
  quoting expressions, 61
  quoting with, 29–30
  in regular expressions, 86
  in replacement strings, 88
  string manipulation, 62
  use of, 12–13
  using in variable names, 45
{*}
  argument expansion, 32
  substitutions, 36

{*m,* } in regular expressions, 81
{*m,n*} in regular expressions, 81
{*m*} in regular expressions, 81
|
  operators and precedence, 56, 57
  in regular expressions, 84
||
  expr syntax, 5–6
  operators and precedence, 56, 57
~
  in file names, 176
  operators and precedence, 55
+
  operators and precedence, 55–56
  as quantifier, 84
  in regular expressions, 81
< in expressions, 55, 56–57
<< in expressions, 55, 57
<=
  expr syntax, 5–6
  precedence and, 55, 56–57
!= in expressions, 55, 56–57
== in expressions, 55, 56–57
>
  expr syntax, 5–6
  precedence and, 55, 56–57
>= in expressions, 55, 56–57
>> in expressions, 55, 57
;
  in basic syntax, 5
  as separator, 21
"
  vs. braces, 12–13
  quoting with, 28–29
  string expressions, 62
∅ in notation, xxxiv
/
  in glob commands, 180
  precedence and, 55–56
2> in redirection syntax, 201
2>@1 in redirection syntax, 201
-
  expr syntax, 5–6
  in gridder, 396
  precedence and, 55–56
! and precedence, 55–56, 57
?: and precedence, 56, 58
?
  glob-style pattern matching, 79
  as quantifier, 84
  in regular expressions, 81
--
  odd file names, 185
  in regexp command, 87
.
  regular expression atoms, 80
  in widget names, 7, 294–295
\
  back references, 85
  backslash substitution, 26–27
  invoking substitution, 11
  in regular expressions, 80–82
  in replacement strings, 88
⇒
  basic syntax, 5
  notation, xxxiv

A
\a in substitution, 27
Aborting commands, 219
abs function, 58
Absolute paths, 178
Absolute qualified names, 161
Access
  file. *See* File access.
  positions, 191–192
  safe interpreter, 277–279
Accessing Tcl variables
  functions, 551–553
  linking with C variables, 556–558
  multiple traces, 561
  reading, 555
  setting and unsetting traces, 558–559
  setting values, 553–555
  trace callbacks, 559–561
  unset callbacks, 562
  unsetting, 556
  whole-array traces, 561
aclocal.m4 file, 710–711
acos function, 58
Action-oriented programming, 525–526
Actions
  defined, 300
  logical, 423–424

`active` state
  for button widgets, 313
  common widget options, 339
  themed widgets, 358
ActiveState Software, 718, 722–724
ActiveTcl, 718
`add` action, 328–329
Addition operator
  precedence and, 55–56
  as quantifier, 84
  in regular expressions, 81
Advanced regular expressions (AREs), 80
`after cancel` command, 248
`after` command, 247, 253–254
`after idle` command, 248
`-after` option
  configuring packer, 398
  packing order, 404
  panedwindows, 336
Aliases
  command, 276–277
  safe interpreter, 278
Aligning text, 309
`all` binding, 423
`all` tag, 439
Allocation, memory, 544
`-alpha` attribute, 487, 488
`alternate` state, 358
`-anchor` option, 398, 403–404
Anchor position, 340–341
AND operator, 56, 57
Anonymous procedures, 151–153
ANSI C. *See* C.
Apartment threading models, 727
Appearance
  themed widget states, 358–360
  themed widget styles, 360–364
`append` command
  defined, 47–49
  working with variables, 39
Applications
  adding Tcl to, 594
  option database entries, 507
  sending commands to Tcl programs, 214–217
  simple Tcl interpreters, 530–531
  Tk, 296

`apply` command
  creating anonymous procedures, 151–153
  defined, 143–144
`arc` type, 434
AREs (advanced regular expressions), 80
`argc` variable, 49
`args` variable, 147–148
Arguments
  applying anonymous procedures, 151–153
  `clientData`, 572–575
  command evaluation, 22–24
  control structures, 12–13
  defaults and variable numbers of, 146–148
  defined, 4
  event binding, 15–18
  expanding, 30–32
  notation, xxxiv
  passing lists as, 30
  in windowing commands, 7–8
`argv` variable, 49
`argv0` variable, 49
Arithmetic
  operators, 64
  performing clock, 258–259
`array` command, 46–47
`array exists` command, 40, 47
`array get` command, 40
`array names` command, 40, 46–47
`array set` command, 40
`array size` command, 40, 46
`array unset` command, 40, 49
Arrays, 19
  accessing with `upvar`, 149
  byte, 540
  defined, 42–43
  vs. dictionaries, 115, 119–120
  `file stat`, 184
  hash table keys, 632
  multidimensional, 45–46
  querying elements of, 45–46
  `tk::Priv`, 518
  tracing variables, 268–269
  whole-array traces, 561
Arrows in scrollbars, 318
`asin` function, 58
Aspect ratios, 482

Associative arrays, 42. *See also* Arrays.
Asynchronous events, 690–694
Asynchronous subprocesses, 211–212
`atan` function, 58
`atan2` function, 58
`atime` option, 184–185
Atoms, 80–83
Attributes
  channel functions, 654–655
  system-specific window, 487–488
`auto_execok` command, 274
`auto_mkindex` command, 230, 233–234
`auto_path` variable, 234
Autoexec, 273
Autoloading
  default version of `unknown`, 273
  defined, 229, 233–234

B
`\b`, 27
Back references, 85
Background errors, 416
`-background` option
  `button` command, 7–8
  configuring bitmap images, 376
  formatting text tags, 455
  for images, 311
  widget color, 307–308
Background processes, 193–194, 685
`background` state, 358
Backslash substitution
  defined, 11
  inside braces, 29
  inside double quotes, 28–29
  more on, 36–37
  syntax overview, 26–27
Backslashes
  back references, 85
  in file names, 176
  in lists, 104–105
  in regular expressions, 80–82
  in replacement strings, 88
Backspaces
  escape sequences, 82
  keysyms, 415
  in substitutions, 27
Basic regular expressions (BREs), 80
`-before` option
  configuring packer, 398
  packing order, 404
  panedwindows, 336
Behavior of themed widgets, 358–360
`bell` command, 518
Bézier curves, 434
`-bgstipple` option, 455
Bignum types, 538
Binary files, 190–191
`binary format` command, 65, 95–97
Binary numeric operands, 54
`binary scan` command, 65–66, 96–99
Binary strings, 95–99
`bind` command
  defined, 413–414
  overview, 416–417
Bindings
  additional uses, 431
  `bind` command, 416–417
  canvas widget, 439–444
  conflict resolution, 421–422
  defined, 413
  event-binding hierarchy, 422–423
  event-driven channel interaction, 206
  events, 15–18
  generating events, 427–428
  keyboard shortcut, 332–333
  named virtual events, 425–426
  text tag, 457–458
  text widget, 449–450
  treeview item tags, 356–357
  widget default, 304–305
Bindtags, 422–423
`bindtags` command, 414, 422
`-bitmap` option, 311
`bitmap` type, 434
Bitmaps
  `image` command, 375–377
  in window decoration, 485–486
Bit-wise operators, 56, 57
`blink` procedure, 254
Blocking channels
  defined, 204–205
  interacting with, 206
Book resources, 732–733
`bool` function, 58

Boolean operators
  expr syntax, 6
  precedence and, 57
Boolean types, 538
-borderwidth option
  formatting text tags, 455
  for frames, 305
Bounds, 84
Braces
  basic dictionary structure, 118
  in glob commands, 180
  inside comments, 33–34
  in lists, 104
  quoting expressions, 61
  quoting with, 29–30
  in regular expressions, 86
  in replacement strings, 88
  string manipulation, 62
  use of, 12–13
  using in variable names, 45
Brackets
  command substitution, 26
  invoking substitution, 11
Branches, 84–85
break command
  defined, 131
  event-binding hierarchy and, 423
  generating exceptions, 224
  loop control, 138
BREs (basic regular expressions), 80
Buffering
  channel buffering mode, 205
  file output, 188
  I/O to and from command pipeline, 203
Building
  embedded Tcl, 714
  Tcl and Tk, 703–707
  TEA, 707–714
  threaded Tcl, 697
Butenhof, David, 695
button command, 6–8
Button widgets
  defined, 294
  event bindings, 16–18
  overview, 312–315
ButtonPress event, 415
ButtonRelease event, 415
Byte arrays, 540

C
C
  accessing Tcl variables from. *See* Accessing Tcl variables.
  analogous Tcl channel commands, 658
  character set issues, 89
  command evaluation, 24
  creating commands. *See* Command creation.
  embedding Tcl. *See* Embedding Tcl.
  evaluating Tcl code in. *See* Evaluating Tcl code.
  expr syntax, 5–6
  fetching values from Tcl objects, 538–539
  file system interaction functions, 681–683
  generating strings with format, 74–75
  integration with Tcl. *See* Tcl and C integration.
  interfaces, 19
  loading libraries, 232
  operators, 5–6
  operators and precedence, 56
  parsing strings with scan, 76–77
  real operand specification, 54
  string functions in UTF-8 and Unicode, 626
  vs. Tcl, 14, 523–524
  Tcl interpreters. *See* Interpreters.
  writing extensions in. *See* Extensions.
c suffix, 17
\c in regular expressions, 81
Call by reference, 148–150
Callbacks
  applying anonymous procedures, 152–153
  channel event handling, 673–676
  clientData and deletion, 572–575
  handling idle, 677–678
  handling with namespace code, 160–161
  idle, 253, 671
  invoking event dispatcher, 678–680
  timer event handling, 676–677
  trace, 559–561
  unset, 562
canvas command, 434–435
Canvas widget
  bindings, 439–444
  canvas scrolling, 444–445
  items and types, 433–436

Canvas widget (*continued*)
  manipulating items with identifiers and tags, 436–439
  PostScript generation, 445
Carriage return in substitutions, 27
`cascade` entry, 328
Cascaded menus, 331–332
Case conversion, 70
Case-sensitivity of variable names, 41
Catalogs, message, 91–95
`catch` command
  defined, 219–220
  generating exceptions, 224–226
  limiting interpreters, 280–281
  return options dictionary, 604
  trapping errors, 222–223
Cavities
  defined, 361
  packing, 396–397
`cd` command, 173, 179
`ceil` function, 58
Cells in gridder, 389
`cget` widget command, 505, 512
`chan close` command, 174
`chan configure` command
  configuring channel options, 204–205
  defined, 175, 198–199
`chan copy` command, 176, 192–194
`chan eof` command, 174, 191–192
`chan event` command, 199, 207–209
`chan flush` command, 174
`chan gets` command, 174
`chan puts` command, 175
`chan read` command, 175
`chan seek` command, 175, 191–192
`chan tell` command, 175, 191–192
Channel events, 206
Channel handlers, 673–676
Channels
  basic file, 186–188
  character encodings and, 90
  configuring options, 204–205
  creating new types, 661–669
  event handling, 672–676
  event-driven interaction, 206–208
  functions, 649–656
  operating system process utilities, 687–689
  operations, 656–658
  registering, 658–660
  standard, 660–661
  transferring between interpreters, 279–280
`char` functions, 536–537
Character classes, 82–83
Character encodings
  defined, 89–91
  working with string utilities, 624–625
  working with text files, 190
Characters
  extracting with `string`, 69
  generating strings with `format`, 74–76
  glob-style pattern matching, 78–79
  handling Unicode and UTF-8 strings, 625–627
  parsing strings with `scan`, 76–78
  in regular expressions, 80–83
  relative placement in gridder, 395–396
  separators, 7
  set issues, 89–91
  split, 112
  substitutions. *See* Substitutions.
  supported by `string is` , 73–74
  text tags, 453–458
  time and date format groups, 254–257
  variables and substitutions, 11
  working with text files, 190
[*chars*]
  glob-style pattern matching, 79
  in regular expressions, 81
`checkbutton` entry, 328
Checkbuttons, 313–314
Child process reaping, 689–690
Choice operator, 56, 58
`chooseDirectory` procedure, 338
Class commands
  creating and destroying widgets, 297–298
  for widgets, 7–8
`class widget` command, 505
Classes
  bindings, 416
  character in regular expressions, 82–83
  characters supported by `string is`, 73–74
  namespaces and, 171–172
  option database entries, 507–508
  themed widget vs. classic widget, 344–345

widget, 7–8, 294, 305. *See also* Widgets.
widget commands, 18, 299–300
Cleanup, process in C, 693–694
Clearing, selection, 470–471
Client communication sockets, 210–212
`clientData` argument, 572–575
`clipboard append` command, 468–469, 474
`clipboard clear` command, 468
`clipboard` command, 473–474
`clipboard get` command, 469, 473
Clipboard model
defined, 467
selection, retrievals, and types, 469–470
`CLIPBOARD` selection, 469
`clock add` command, 248, 258–259
`clock` command, 254–259
`clock format` command, 248, 254–257
`clock microseconds` command, 248, 254
`clock milliseconds` command, 248, 254
`clock scan` command, 248, 257–258
`clock seconds` command, 248, 254
Closing
command pipeline, 204
windows, 489–490
Code evaluation. *See* Evaluating Tcl code.
Collections. *See* Lists, Dictionaries.
Colon in qualified names, 161–162
Colors for widgets, 307–308
Columns
gridder options, 389–390
managing treeview, 353–355
spanning gridder, 393–394
Combobox widget, 345–346
Command creation
`clientData` and deletion callbacks, 572–575
command functions, 565–567
deleting, 575
fetching and setting parameters, 576–578
functions, 564–565
how procedures work, 578–579
registration, 567–569
result protocol, 569
`Tcl_AppendResult`, 569–570
`Tcl_SetResult` and `interp->result`, 570–572
traces, 579
`command` entry, 327
Command functions, 565–567
`-command` option
`button` command, 7–8
widget commands, 18
Command procedures. *See* Procedures.
Command substitution
defined, 11
syntax overview, 26
Commands
additional Tk, 513–514
analogous C functions, 658
`canvas`, 434–435
configuration option, 505–506
control flow, 131–132
creating and destroying widgets, 297–298
creating lists as, 113–114
dictionary, 116–118
errors and exceptions, 219–220
evaluation, 22–24
event binding, 15–18
expression, 53
file access, 173–176
file system interaction, 681–683
for fonts, bitmaps and images, 367–369
geometry manager, 386
history, 283–284
information about, 262–263
inspecting namespaces, 163–164
for interconnection, 300–301
library, 230–231
list, 101–103
namespace, 155–158
normal and exceptional returns, 35
notation, xxxiv
`peer` text widget, 463–465
process, 197–199
re-executing from history list, 285–286
script files, 8–10
selection and clipboard, 468–469
sending to Tcl programs, 214–217
string manipulation, 65–69
string utility completeness, 627–628
syntax overview, 21–22
Tcl as language, 14–15
Tcl internals, 247–253
`tclsh` application, 4–6
Tcl/Tk benefits, xxxii–xxxiii
Tk windowing, 6–8

Commands (*continued*)
  variables, 39–40
  variables and substitutions, 11
  widget, 18, 299–300
  window management, 478–481
  working with ensembles, 164–169
Comments
  in switch statements, 136
  syntax overview, 32–35
Communication
  commands for widget interconnection, 300–301
  interapplication, 19
  TCP/IP socket, 210–214
Comparisons
  operators, 64
  string, 71
  text widget index, 452
Compatibility issues in releases, 717–718
comp.lang.tcl newsgroup, 732
Completeness, command, 627–628
Completion codes, 600–603
Composite objects, 540
-compound option
  defined, 311
  in themed widgets, 365
concat command, 101, 105–106
Condition variables, 699–701
Conditions, 699–701
Configuration options
  bitmap image, 376–377
  cget widget command, 512
  chan configure, 204–205
  color, 307–308
  commands, 505–506
  common themed widget, 364–365
  common widget, 339–342
  configure widget command, 511–512
  creating and destroying widgets, 298
  database, 506–507
  database entries, 507–508
  ensembles, 166–167
  for frames, 306–307
  geometry manager, 408–409
  gridder, 390–392
  for labels, 309–312
  named font, 370–371
  notebook, 347–348
  option command, 510–511
  packer, 398–399
  panedwindows, 336–337
  photo images, 377–379
  priorities in database, 509–510
  RESOURCE_MANAGER property and .Xdefaults file, 508–509
  text widget tags, 454–456
  themed widget style, 362–364
  treeview columns and headings, 355
  treeview item selection management, 355
  treeview item tags, 356
  treeview items, 351
  widget, 294, 304
  windowing commands, 7
configure command
  defined, 18
  TEA standard options, 707–709
Configure event, 415
configure widget command, 506, 511–512
configure.in file, 711–714
Conflicts
  in event bindings, 421–422
  priorities in option database, 509–510
Console window, 6
Constraints, 80
Contents
  copying file, 192–194
  listing directory, 179–181
  PostScript generation, 445
continue command
  defined, 131
  generating exceptions, 224
  loop control, 138
Contributed extensions to Tcl/Tk, 721–724
Control flow
  commands, 131–132
  eval, 139–140
  executing from files, 140–141
  if, 132–133
  loop control, 138
  looping commands, 136–138
  switch, 133–136
Control structures
  additional features, 19
  creating new, 150–151
  Tcl, 12–14
Conversion specifiers, 94–95

Conversions
  character encodings, 89–91
  generating strings with `format`, 74–76
  parsing strings with `scan`, 76–78
  between strings and lists, 111–112
  types and, 63–64
  working with character encodings, 624–625
Copying
  with `clipboard`, 473–474
  file content, 192–194
  files, 182–183
Core commands, xxxi
`cos` function, 58
`cosh` function, 58
`countdown` procedure, 34–35
Counting, reference, 540–541
Current event number, 287
`current` tag, 439
Current working directory, 179
Cursors
  common widget options, 341–342
  insertion, 323
  themed widget options, 364
Customization
  `aclocal.m4` file, 710–711
  channel instance, 662
  `configure.in` file, 711–714
  dialog, 499–504
  `Makefile.in` file, 714
Cutting with `clipboard`, 473–474

D
Dashes in odd file names, 185
`-data` option
  configuring bitmap images, 376
  configuring photo images, 377
Data storage, 41–42
Databases, 728
Databases, option. *See* Option database.
Date manipulation, 254–259
Decimal operands, 54
Decorating windows, 485–486
Default bindings, 304–305
Default fonts, 370
Default installation repositories, 722–724
Defaults and variable number of arguments, 146–148
Deferred evaluation, 29–30
Deiconified state
  creating custom dialogs, 500
  defined, 483–484
Delayed operations, 514–515
Delaying for time, 253–254
Deleting
  `clientData` and deletion callbacks, 572–575
  commands, 269, 575
  destroying widgets, 514
  with `dict unset`, 127
  files, 181–182
  hash table entries, 637
  hash tables, 632–633
  interpreters, 531
  with `lreplace`, 107
  named fonts, 372
  namespace, 159
  with `namespace forget`, 162
  thread termination, 698
  traces, 562
  tracing commands, 270
  treeview items, 353
  with `unset` and `array unset`, 49
  widgets, 298
  windows, 490–491
Demo script for widgets, 305
Descriptions, font, 374–375
Desktop environment, 293
`destroy` command
  defined, 298
  destroying widgets, 513, 514
`Destroy` event, 415
Detail field
  for event patterns, 417–418
  for key and button events, 415
Advantages, Tcl, xxxii
Dialogs
  custom, 499–504
  modal interactions, 495–499
  standard, 337–339
`dict append` command, 123–124
`dict` command, 116, 123–126
`dict create` command, 116, 120–121
`dict exists` command, 116, 122–123
`dict filter` command, 116
`dict for` command, 116, 122–123

dict get command
  basic dictionary structure, 118–120
  defined, 116
  working with nested dictionaries, 126–127
dict incr command, 116, 124
dict keys command, 117, 122–123
dict lappend command, 117, 124
dict merge command, 117, 121
dict remove command, 117
dict replace command, 117, 120–121
dict set command
  defined, 117
  updating dictionaries, 121
  working with nested dictionaries, 126–127
dict size command, 117, 122–123
dict unset command
  defined, 117
  updating dictionaries, 121
  updating dictionary values, 125
  working with nested dictionaries, 126–127
dict update command, 117, 124–125
dict values command, 117
dict with command, 118, 126–129
Dictionaries
  basic structure, 117–120
  commands, 116–118
  converting betweens arrays and, 47
  creating and updating, 120–121
  defined, 19
  examining, 122–123
  managing return options, 604–605
  overview, 115–116
  return options, 225–226
  string map command, 72
  updating values, 123–126
  working with nested, 126–129
Dictionary objects
  defined, 644–648
  functions, 639–642
Diekhans, Mark, 730
Directories
  creating, 181
  current working, 179
  layout for TEA extensions, 709–710
  libraries. *See* Libraries.
  listing contents, 179–181
  name manipulation, 176–179
-disabled attribute, 488
disabled state, 358
Disk files
  handling channel events and, 675–676
  working with, 181–185
Dispatcher, event, 678–680
Display lists, 435
Displays, 292
DLL (Dynamic Link Library), 232
DND (Drag and Drop), 474–475
do command
  creating with uplevel, 150–151
  generating exceptions, 226–227
Dockable windows, 488–489
Dollar sign
  command evaluation, 24
  invoking substitution, 11
  variable substitution, 24–25, 43–45
Double colon, 161–162
double function, 58
Double modifier, 418
Double quotes, 28–29
Double types, 538
Drag and Drop (DND), 474–475
Duquette, William, 726
Dynamic languages, xxxi
Dynamic Link Library (DLL), 232
Dynamic strings, 617–621
Dynamic Tcl objects, 539
Dynamic Tcl scripts, 548

E
Echo servers, 212–214
Elements
  defined, 101
  examining dictionaries, 122–123
  extracting from lists, 109
  inserting and deleting from lists, 644
  themed widget style, 361–362
Elements, array
  defined, 42–43
  multidimensional arrays, 45–46
  querying, 45–46
  removing, 49
-elide, 454
Ellipses in canvas widgets, 433
else clauses, 133
elseif clauses, 133
Embeddable commands languages, xxxi

Embedded images, 460–462
Embedded windows, 459–460
Embedding Tcl
  adding to application, 594
  building, 714
  in C, 522
  creating shells, 596–597
  functions, 593
  initialization, 595
Encapsulated PostScript, 445
`encoding convertfrom` command, 66
`encoding convertto` command, 66
`encoding names` command
  character set issues, 89
  defined, 66
`encoding system` command, 66
Encodings, character
  defined, 89–91
  working with string utilities, 624–625
  working with text files, 190
End-of-line conventions, 188–190
Ensembles
  defined, 23
  working with namespaces and, 164–169
`Enter` event, 415
`enter` traces, 271–272
`enterstep` traces, 271–272
`entier` function, 64
Entries
  creating hash table, 633–634, 635
  defined, 16
  deleting hash table, 637
  in listboxes, 317–318
  option database, 507–508
  overview, 323–327
`env` variable
  defined, 50
  processes and, 210
  tracing, 268–269
Epoch time
  defined, 184–185
  manipulation, 254–259
`eq` operator, 55
Equals sign, 55, 56–57
EREs (extended regular expressions), 80
`error` command, 220, 222
Error messages, 221
`errorCode` variable
  defined, 603–604
  setting, 221–222
`errorInfo` variable
  adding to stack trace in, 605–608
  defined, 35
Errors
  background, 416
  background and `bgerror`, 227–228
  commands, 219–220
  entry validation, 326–327
  exceptional returns, 35
  exceptions, 223–227. *See also* Exceptions.
  generating from Tcl scripts, 222
  raised by `exec`, 202
  in system calls, 196
  trapping, 222–223
  unknown commands, 274
  what happens after?, 220–222
Escape sequences
  in file names, 176
  regular expression character class, 83
  in regular expressions, 82
`eval` command
  control flow, 139–140
  defined, 131
Evaluating Tcl code
  dynamically building scripts, 548
  embedding Tcl, 595
  functions, 545–546
  in namespaces, 158–161
  overview, 546–547
  Tcl expressions, 549
Evaluation
  command, 22–24
  deferring, 29–30
  nested script, 31
`event add` command, 414
`event delete` command, 414
Event dispatcher, 678–680
`event generate` command, 414, 427–428
Event handlers
  background errors, 227–228
  commands for interconnection, 300
  defined, 206
  registering, 207–209
  Tk, 297

Event handling
  asynchronous, 690–694
  channel events, 672–676
  event dispatcher, 678–680
  functions, 671–672
  idle callbacks, 677–678
  timer events, 676–677
`event info` command, 414
Event loops
  defined, 297
  entering Tcl with `vwait`, 206–207
  processing times, 423–424
Event patterns, 417–419
`!event` shortcut, 286
Event specifiers, 17
Event-driven programming
  `after` command, 253–254
  Tk applications, 297
Events
  `bind`, 416–417
  binding, 15–18
  commands, 413–414
  conflict resolution, 421–422
  current event number, 287
  event-binding hierarchy, 422–423
  event-driven channel interaction, 206–208
  generating, 427–428
  history list, 284–285
  logical actions, 428–431
  named virtual, 425–426
  overview, 414–416
  processing times, 423–424
  sequences, 419
  specifying, 285
  substitutions in scripts, 419–421
  tag bindings, 457–458
  Tk, 297
  in windowing systems, 293
Examples
  `blink`, 253–254
  `do`, 150–151, 226–227
  `fac`, 145
  `factorial`, 15–18
  `flash`, 515
  `GetGizmo`, 635
  `GizmoCreateCmd`, 633–634
  `GizmoDeleteCmd`, 637
  `GizmoSearchCmd`, 636
  `GizmoStatCmd`, 638
  `GizmoTwistCmd`, 635
  `graphs`, 440–441
  `Hello, World!`, 6–8
  `inc`, 417, 578–579
  `map`, 618
  `plus`, 144–145
  `power`, 167
  `printargs`, 50
  `printArray`, 149
  `printVars`, 147
  `ruler`, 435–436
  `sum`, 148
  `waitWindows`, 499
Exceptional returns, 35
Exceptions
  adding to stack trace in `errorInfo`, 605–608
  completion codes, 600–603
  functions, 599–600
  managing return options dictionary, 604–605
  overview, 223–227
  setting `errorCode`, 603–604
  `Tcl_Panic`, 608–609
Exclamation points, 55–56
`exec` command
  accessing process IDs, 209
  autoexec, 273
  defined, 197
  invoking subprocesses with, 199–203
Executable files, 184
Executable scripts, 9
Execution
  defined, 22–24
  from files, 140–141
  re-executing commands from history list, 285–286
  timing command, 259
  tracing command, 271, 579
`exit` command
  defined, 6, 198
  process termination, 199
Exiting
  asynchronous event handlers, 693–694
  threads, 698
`exp` function, 58

-expand option in packer, 398, 401–403
Expansion
  argument, 30–32
  configuring packer, 401–403
  file name and exec, 202–203
Expect, 729–730
Explicit focus models, 493–494
Exponentiation operator, 55–56
Exporting namespace commands, 162
expr command
  defined, 53
  evaluating, 23
  getting started, 4
Expressions
  commands, 53
  evaluating from C, 549
  list manipulation, 63
  matching regular, 622–623
  math functions, 58–60
  numeric operands, 54
  operators and precedence, 55–58
  Pattern matching with, 79–87
  precision, 64
  string manipulation, 62–63
  substitutions, 60–61
  Substitutions with, 87–88
  Tcl as language, 14–15
  types and conversions, 63–64
Extended regular expressions (EREs), 80
Extended Tcl (TclX), 730
Extensible languages, xxxi
Extensions
  connected C to Tcl, 524–525
  database programming, 728
  defined, xxxi
  file and directory name manipulation, 178
  functions, 581–582
  ifconfig, 585–592
  Init, 582–583
  integrating Tcl and C, 522
  namespaces, 584
  obtaining and installing, 721–724
  packages, 583–584
  Snack, 725–726
  Tcl stubs, 584–585
  Tcl/Tk benefits, xxxii
  TEA, 707–714
  Thread, 727
  TkCon, 724
  virtual file systems, 194–196
  XML programming, 727–728
External padding in packer, 404
Extraction
  character, 69
  element, 109

F
\f in backslash substitution, 27
fac example, 145
factorial procedure, 12–14, 15–18
-failindex option, 73
False values, 57–58
fconfigure command, 90
Fetching command parameters, 576–578
-fgstipple option, 455
File access
  additional features, 19
  commands, 173–176
  current working directory, 179
  on disks, 181–185
  errors in system calls, 196
  listing directory contents, 179–181
  name manipulation, 176–179
  reading and writing files, 185–194
  virtual file systems, 194–196
file command, 183–185
file copy command, 182–183
file delete command, 181–182
file dirname command, 178
File events
  defined, 206
  registering handlers, 207–209
file extension command, 178
File extensions, 9–10
File handlers, 673–676
File identifiers, 187
file join command, 176–177
file mkdir command, 181
File name expansion, 202–203
file nativename command, 177
file normalize command, 178
-file option
  configuring bitmap images, 376
  configuring photo images, 377
file option command, 174

file pathtype command, 178–179
file rename command, 183
file rootname command, 178
file split command, 176–177
File system interaction
  Tcl functions, 681–683
  virtual file systems, 683
file tail command, 178
file volumes command, 179
FILE_NAME type, 469–470
Files
  executing from, 140–141
  navigation dialogs, 338–339
  script, 8–10, 263
-fill option
  configuring packer, 398, 400–401
  vs. -expand, 403
Flags
  operating system process, 687–688
  reading variable, 555
  Tcl variable values, 554
  variable linking, 557
flash example, 515
floor function, 58
Flow, control. *See* Control flow.
flush command, 188
fmod function, 58
Focus, input
  commands, 491–492
  overview, 493–495
focus command, 491–492, 494
focus state, 358
Focus widgets, 301
FocusIn event, 415
FocusOut event, 415
font actual command, 367
font configure command, 368, 371
font create command, 368, 370
font delete command, 368, 372
font families command, 368, 373–374
font measure command, 368, 374
font metrics command, 368
font names command, 368, 372
-font option
  in button command, 7–8
  for formatting text tags, 455
  for labels, 310
Fonts
  command overview, 367–368
  font command, 369–375
for command
  defined, 131–132
  looping, 136–138
forAllMatches procedure, 451–452
foreach command
  defined, 132
  looping, 136–138
  using with array names, 47
-foreground option
  for bitmap image configuration, 376
  in button command, 7–8
  defined, 308
  for formatting text tags, 455
  for images, 311
Form feeds
  escape sequences, 82
  in substitution, 27
format command
  defined, 66
  generating strings with, 74–76
Format groups
  clock format, 254–257
  clock scan, 257
-format option in photo images, 377
Formats
  additional image with Img, 725
  font, 374–375
  representing information, 526
  text tags, 453–458
  types for binary strings, 96–97
Forward slash
  in glob commands, 180
  precedence and, 55–56
Frames
  defined, 306–307
  hierarchical geometry management with, 405–407
  in panedwindows, 335
  window decorations, 485–486
-fullscreen attribute, 487
Functions
  for accessing Tcl variables, 551–553
  analogous Tcl channel commands, 658
  channel, 649–656

command creation, 564–565
for evaluating Tcl code, 545–546
event handling, 671–672
exception, 599–600
file system interaction, 681–683
hash table, 630–631
`Init`, 582–583
interpreter, 527–529
manipulating list and dictionary objects, 639–642
math, 58–60
operating system, 685–687
string utility, 611–617
Tcl embedding, 593
Tcl extension, 581–582
Tcl object, 534–537
thread management, 695–697

G
`-gamma` option in photo images, 377
Geometry managers
commands, 386
defined, 8
gridder, 389–396
hierarchy, 405–407
options, 408–409
other in Tk, 409–411
overview, 385, 387–389
packer, 396–404
padding, 404–405
placer, 405
text widget as, 459
Tk, 298–299
widget stacking order, 407–408
window management. *See* Window managers.
Geometry propagation, 409
`GetGizmo` example, 635
`getOpenFile` procedure, 338
`gets` command, 203
`getSaveFile` procedure, 338
`GizmoCreateCmd` example, 633–634
`GizmoDeleteCmd` example, 637
Gizmos
checking statistics, 638
creating, 633–634
defined, 629–630
searching, 636
`GizmoSearchCmd` example, 636
`GizmoStatCmd` example, 638
`GizmoTwistCmd` example, 635
`glob` command
defined, 173–174
listing directory contents, 179–181
`global` command, 51, 144
Global grabs, 496–497
Global namespaces, 91–95
Global variables
accessing with `upvar`, 149–150
defined, 13–14
managed by Tk, 517–518
overview, 49–50
procedures, 146
Glob-style pattern matching
listing directory contents, 179–181
String manipulation, 78–79
string matching, 621–622
Glue language, xxxii
gm command, 386, 408–409
gm `configure` command, 386, 408
gm `forget` command, 386, 408–409
gm `info` command, 386, 408
gm `slaves` command, 386, 408
GNU Autoconf
building Tcl and Tk, 703–707
TEA, 707–714
`grab` command, 492, 495–497
Grabs
defined, 301
modal interactions, 495–497
Grammar, 14–15
Graphical user interface (GUI) creation with Tk. *See* Tk.
`graphs` example, 440–441
Gravity, mark, 450
Greater than, 55, 56–57
Greater than or equal to, 55, 56–57
Greedy quantifiers, 84–85
`grid` command
event binding, 17
"Hello, World!" example, 6–8
introduction to widgets, 304
Grid managers, 298–299
Gridded geometry management, 483
Gridded windows, 483–484
Gridders, 8, 385, 389–396

Groups, window, 486–487
GUI (Graphical user interface) creation with Tk. *See* Tk.

H
Handlers, event. *See* Event handlers.
Handling events. *See* Event handling.
Hash tables
- creating and deleting, 632–633
- creating entries, 633–634
- deleting entries, 637
- finding existing entries, 635
- functions, 630–631
- keys and values, 631–632
- overview, 629–630
- searching, 636–637
- statistics, 638

Hashmarks in comments, 32–35
Headings, treeview, 353–355
`-height` option
- for configuring photo images, 377
- in panedwindows, 336
- in widgets, 339–340

"Hello, World!" example, 6–8
Hexadecimal operands, 54
Hidden commands and slave interpreters, 277–279
`-hide` option in panedwindows, 336
Hiding text, 325
Hierarchies
- event-binding, 422–423
- geometry management, 405–407
- widget, 7–8

History
- commands, 283–284
- current event number, 287
- event specification, 285
- history list, 284–285
- re-executing commands from list, 285–286
- shortcuts implemented by `unknown`, 286–287

`history clear` command, 282
`history` command, 282
`history info` command, 282
`history keep` command, 282
History lists
- defined, 284–285
- re-executing commands from, 284–285

`history nextid` command, 282, 287
`history redo` command, 282, 285–286
Hobbs, Jeff, 724
Hostnames, 694
Human-readable time and data
- generating, 254–257
- scanning, 257–258

`hypot` function, 59

I
`i` suffix, 17
Icon decorations, 485–486
Iconified state, 483–484
Identifiers
- accessing process, 209–210
- file, 187
- manipulating items with canvas widget, 436–439
- process, 208–209
- retrieving thread, 701
- tab in notebook widgets, 347

Idle callbacks
- defined, 671
- handling, 677–678
- time delay commands, 253

Idleness, 514
IEEE Floating Point Standard, 63
`if` command
- comments, 34–35
- control flow, 132–133
- defined, 53, 132

`ifconfig` command, 585–592
`image create` command, 311, 368, 376
`image delete` command, 368
`image height` command, 368
`image inuse` command, 369
`image names` command, 369
Image objects, 375
`-image` option in themed widgets, 365
`image` type, 434
`image type` command, 369
`image types` command, 369
`image width` command, 369
Images
- additional formats, 725
- commands, 368–369

embedded in text widget, 460–462
image command, 375–383
widget options, 311
Img format, 725
Immediate values, 201
Implicit focus models, 493–494
Importing
namespace commands, 162
variables from other namespaces, 169
in operator
list manipulation, 63
operators and precedence, 56
-in option
configuring gridder, 390
configuring packer, 398
inc example, 417, 578–579
incr command
defined, 47–49
working with variables, 39
Indices
in basic list structure, 103–105
building with autoloader, 233–234
extracting characters, 69
text widget, 450–451
treeview item, 353
info args command, 248, 261–262
info body command, 248, 261–262
info cmdcount command, 248, 263
info command, 259–264
info commands command, 249, 262–263
info complete command, 249, 263
info default command, 249, 261–262
info exists command, 249, 260
info globals command, 249, 260
info hostname command, 249, 264
info level command, 249, 262
info library command, 249, 264
defined, 230
info loaded command, 230
info locals command, 249–250, 260
info nameofexecutable command, 250, 264
info patchlevel command, 250, 264
info procs command, 250, 261–262
info script command, 250, 263
info sharedlibextension command
defined, 230, 250
loading libraries, 232
runtime information, 264
info tclversion command, 250, 264
info vars command, 250, 260–261
Init function, 582–583
Initialization scripts
defined, 297
Tcl embedding, 595
Input focus
commands, 491–492
overview, 493–495
Input/output (I/O). *See* I/O (input/output).
insert action, 323
Insertion cursors, 323
Installation
package, 238
Tcl and Tk, 717–719
Tcl extensions, 721–724
Tcl module, 240–241
Tclkit, 243
Installation repositories, 722–724
instate widget command, 359–360
int function
defined, 59
types and conversions, 64
Integers
clock arithmetic, 258–259
generating strings with format, 74–76
numeric operands, 54
numerical object types, 537–538
types and conversions, 63–64
Integrating Tcl and C. *See* Tcl and C integration.
Interactive interpreters, 10
Interactive resizing, 481–482
Interapplication communication, 19
Interconnection commands, 300–301
Interface integration. *See* Tcl and C integration.
Internal padding
in packers, 404
in widgets, 341
Internal windows, 292
Internals, Tcl. *See* Tcl internals.
interp alias command, 250, 276–277
interp bgerror command, 220, 227–228
interp command, 275–276

`interp create` command
  defined, 251
  safe interpreters, 277–278
  slave interpreters, 275
`interp delete` command, 251, 275
`interp eval` command, 251, 275
`interp expose` command, 251, 279
`interp hide` command, 251, 279
`interp invokehidden` command, 251
`interp limit` command, 251
`interp recursionlimit` command, 251, 281
`interp share` command, 251, 280
`interp transfer` command, 251, 280
`interp->result`, 570–572
Interpreters
  autoloading, 233–234
  command evaluation, 22–24
  deleting, 531
  errors and exceptions, 220
  functions, 527–529
  loading libraries into, 232
  managing internals. *See* Tcl internals.
  multiple, 531–532
  overview, 529
  simple Tcl application, 530–531
  slave, 274–281
  Starpacks, 245–246
  Tcl as language, 14–15
  Tclkit, 243
  thread safety, 697
`invalid` state, 359
I/O (input/output)
  basic file, 186–188
  channel event handling, 673–676
  to and from command pipeline, 203–204
  configuring channel options, 204–205
  `exec` support for redirection, 200–201
  transferring channels between interpreters, 279–280
`-ipadx`
  configuring gridder, 391
  configuring packer, 399
  geometry management, 404–405
`-ipady`
  configuring gridder, 391
  configuring packer, 399
  geometry management, 404–405
Itcl, 726
Items
  canvas widget, 433–436
  manipulating with identifiers and tags, 436–439
  treeview, 350–353
  treeview selection management, 355–356
  treeview tags, 256–257

J
Java and Tcl integration, 728–729
`join` command, 102, 111–113
Joinable threads, 697–698
`-justify` option
  in labels, 309
  in text tags, 455

K
`\k` in regular expressions, 81
Kenny, Kevin, 701
Key replacements, 72
Keyboard events and tag bindings, 458
Keyboard shortcuts
  binding to named virtual events, 425–426
  defined, 332–333
Keyboard traversal
  defined, 332
  in notebook widgets, 349
Keyboards and grabs, 497
`KeyPress` events, 414
`KeyRelease` events, 414
Keys
  in basic dictionary structure, 118–119
  defined, 629
  dictionary objects, 644–648
  examining dictionaries, 122–123
  `file stat`, 184
  hash table, 631–632
  input focus, 493–495
  updating dictionary values, 123–126
Keysyms, 415–416

L
Label widgets
  defined, 309–312
  event binding, 16–18
`-labelanchor` option, 312
Labelframes, 312

Language
  creating localized message files, 93–94
  Tcl and C integration. *See* Tcl and C integration.
  Tcl as, 14–15
lappend command, 102, 108–109
lassign command, 102, 109
Layout
  formatting text tags, 455–456
  geometry management, 387–388
  for TEA extensions, 709–710
  themed widget styles, 361
Leaders, window, 486–487
Leave event, 415
leave traces, 271–272
leavestep traces, 271–272
Lehenbauer, Karl, 730
Length
  basic list structure, 104–105
  string, 70
Less than, 55, 56–57
Less than or equal to, 55, 56–57
Libes, Don, 730
Libraries
  autoloading, 233–234
  commands, 230–231
  default version of unknown, 273
  extensions as, 582
  load, 232
  overview, 229
  packages, 234–239
  packaging scripts as Starkits, 242–246
  Tcl modules, 239–241
  Tcllib, 725
  vs. tclsh, 596
  Tk variables, 517–518
  using, 232
Limiting interpreters, 280–281
lindex command
  basic list structure, 103
  defined, 102
  evaluating, 23
Line continuation, 17
line type, 434
Lines in canvas widgets, 434
Linking Tcl variables to C variables, 556–558
linsert command, 102, 107
list command, 102, 105–106
List objects
  defined, 642–644
  functions, 639–642
Listboxes, 317–318
Listening sockets, 212–214
Lists
  vs. arrays, 43
  basic structure, 103–105
  commands, 101–103
  converting between strings and, 111–112
  creating, 105–106
  creating commands as, 113–114
  defined, 19, 23
  directory contents, 179–181
  extracting elements, 109
  history, 284–285
  manipulation expressions, 63
  modifying, 106–109
  packing, 396
  passing as arguments, 30
  re-executing commands from history list, 285–286
  searching, 110
  sorting, 111
llength command, 102, 103–105
-lmargin1 option, 455
-lmargin2 option, 455
load command, 232
Local grabs vs. global grabs, 496–497
Local variables
  accessing with upvar, 149–150
  defined, 13
  procedures, 146
Locale
  generating readable time and date, 256–257
  scanning time and date, 258
  setting msgcat, 91–92
log command, 270–271
log function, 59
log10 function, 59
Logical actions, 428–431
Logical operators, 53
Long types, 538
Looping
  after command, 253
  commands, 136–138

Looping (*continued*)
  control, 138
  with dict for, 123
  entering Tcl event with vwait, 206–207
  generating exceptions, 226–227
lower command, 386, 407–408
lrange command, 102, 106
lrepeat command, 102, 105–106
lreplace command, 102, 107–108
lsearch command, 102–103, 110
lset command, 103, 108–109
lsort command, 103, 111
lstat option, 185

M

\m in regular expressions, 81
\M in regular expressions, 81
m suffix, 17
Mac OS X
  binding to named virtual events, 425–426
  building Tcl and Tk on, 705–706
  cursor options, 342
  executable scripts on, 9
  platform-specific menus, 333
  system-specific window attributes, 488
  tcl_platform array, 50
Main widgets, 7
Main windows, 295
Major releases, 717–718
Major version number, 717
Makefile.in file customization, 714
Managers, geometry. *See* Geometry managers.
Managers, window. *See* Window managers.
Managing Tcl internals. *See* Tcl internals.
Manual entries, xxxiii
map command, 153
Map event, 415
map example, 618
-map option in ensembles, 167
Marks, 450–451
-maskdata option in bitmaps, 376
-maskfile option in bitmaps, 377
Master interpreters, 274
Master widgets, 387–389
Master windows, 486
Matching
  patterns. *See* Pattern matching.
  regular expressions, 622–623
  strings, 621–622
Math functions
  overview, 58–60
  types and conversions, 64
max function, 59
McLennan, Michael, 726
mcload command, 92
mclocale command, 91–93
Memory allocation, 544
Menu bars, 329–331
Menubuttons, 316
Menus, 327–334
Message catalogs, 91–95
Message files
  creating localized, 93–94
  defined, 91
messageBox dialog, 337–338
Messages, error, 221
min function, 59
Minor releases, 717–718
Minor version number, 717
-minsize option in panedwindows, 336
Minus sign, 55–56
Mk4tcl, 728
Modal interactions, 495–499
Modal user interface elements
  defined, 330
  messageBox dialog, 338
-modified attribute, 488
Modified virtual event, 458, 462–463
Modifier keys, 416, 418
Modify flag, 462–463
Module path, 240
Modules
  defined, 229
  working with Tcl, 239–241
Monitoring. *See* Tracing.
monospace notation, xxxiv
Motion events, 415
Mouse
  button widgets, 312–313
  DND, 474–475
  input focus and, 493–494
  modal interactions, 495–499
  pop-up menus, 334
MouseWheel event, 415
msgcat package, 91–95

Multidimensional arrays, 45–46
Multiple interpreters, 531–532
Multiple traces, 561
Multiplication operator, 5, 55–56
Multithreading
  Tcl scripting, 727
  Tk support, 296
Mutexes, 698–699
Mysqltcl, 728

N

`\n` in substitution, 27
Named fonts
  defined, 369–370
  manipulating and using, 370–373
Named keys, 125
Named virtual events, 425–426
Names
  application and using `send`, 216
  array, 42–43
  array commands, 40
  in basic dictionary structure, 118–120
  controlling resolution paths, 170–172
  dealing with odd file, 185
  file name expansion and `exec`, 202–203
  managing image, 375
  manipulating file and directory, 176–179
  manipulating qualified, 161–162
  renaming files, 183
  signal, 693
  simple variables, 41
  tracing array variables, 268
  unknown commands, 272–274
  utility package commands, 238
  variable substitution, 44–45
  widget, 7, 294–295
`namespace children` command, 155–156, 163
`namespace code` command, 156, 160–161
`namespace` command, 51, 158
`namespace current` command, 156, 163
`namespace delete` command, 156, 159
`namespace ensemble` command, 164–169
`namespace ensemble configure` command, 156, 166–167
`namespace ensemble create` command
  controlling configuration, 166–167
  defined, 156, 164–165
`namespace ensemble exists` command, 156, 165
`namespace eval` command, 156, 158–161
`namespace exists` command, 156
`namespace export` command
  creating ensembles, 164
  defined, 156, 162
  inspecting namespaces, 164
`namespace forget` command, 156–157, 162
`namespace import` command, 157, 162
`namespace origin` command, 157, 164
`namespace parent` command, 157, 163
`namespace path` command, 157, 170–172
`namespace qualifiers` command, 157, 161–162
`namespace tail` command, 157, 161
`namespace unknown` command, 157
`namespace upvar` command, 158
`namespace which` command, 158, 163–164
Namespaces
  accessing variables from other, 169–170
  commands, 155–158
  controlling name resolution path, 170–172
  ensemble commands, 164–169
  evaluating Tcl code in, 158–161
  exporting and importing commands, 162
  images and, 383
  inspecting, 163–164
  manipulating qualified names, 161–162
  mapping functions to commands, 59–60
  C language extensions, 584
  Tk widgets, 304
  using message catalogs with, 95
  variables, 14, 146
Natural window size, 481
`ne` operator, 56
Nesting
  with braces, 12, 29
  dictionary, 126–129
  ensembles, 165–166
  handling errors, 221
  in lists, 105
  script evaluation, 31
  slave interpreters, 275
  treeview items, 351
  widgets, 294
Neumann, Gustaf, 726

Newline characters
  backslash substitution, 26–27
  in basic syntax, 21
  variables and substitutions, 11
`\newline whitespace`, 27
Newsgroup for Tcl and Tk, 732
`ni` operator
  list manipulation, 63
  precedence and, 56
`-nocase` option, 71
Non-blocking I/O, 675
Non-capturing subexpressions, 85
Non-greedy quantifiers, 84–85
Normal returns, 35
Normal window state, 483
Not equal, 55, 56–57
NOT operator, 55, 57
Notation, xxxiv
Notebook widget
  defined, 346–349
  as geometry manager, 410–411
`-notify` attribute, 488
Numeric operands, 54
Numerical objects, 537–538

O
Object-oriented programming
  vs. Action-oriented programming, 525–526
  Tcl, 726
Objects. *See* Tcl objects.
Octal operands, 54
Ones complement, 57
Online resources, 731–732
`\ooo` in substitution, 27
`open` command
  command pipeline, 203
  defined, 186–187, 198
`open name` command, 174
Operands
  defined, 53
  list manipulation, 63
  numeric, 54
  string manipulation, 62–63
  substitution, 60–61
  types and conversions, 63–64
Operating system utilities
  asynchronous events, 690–694
  functions, 685–687
  miscellaneous, 694
  processes, 687–689
  reaping child processes, 689–690
Operating systems
  character encodings and, 89–90
  file and directory name manipulation, 176–179
  system-specific window attributes, 487–488
  Tk support, 293
  variable describing, 50
Operations, channel, 656–658
Operators
  defined, 53
  `expr` syntax, 5–6
  overview, 55–58
  types and conversions, 63–64
`option` command, 506, 510–511
Option database
  defined, 506–507
  entries, 507–508
  `option` command, 510–511
  option priorities, 509–510
Options. *See* Configuration options.
OR operator
  `expr` syntax, 5–6
  precedence and, 56, 57
Oratcl, 728
Ordered collections, 115
Orientation
  panedwindows, 334
  progressbar, 349
  scrollbar, 320
  separator, 350
Origins, namespace, 164
Output buffering, 188
`oval` type, 433
Override-redirect windows, 487
`-overstrike` option
  configuring font, 371
  formatting text tags, 455
Ownership, selection
  locating selections, 470–471
  model, 467
  retrievals, types and, 469

P
`p` suffix, 17
`pack` command, 399–400

package ifneeded command, 230, 237
package names command, 230, 238
package prefer command, 230–231
package provide command
  defined, 231, 235–236
  using Tcl modules, 238–239
package require command
  defined, 231, 235
  using Tcl modules, 239
package vcompare command, 231
package versions command, 231, 238
Packages
  defined, 229
  packaging scripts as Starkits, 242–246
  Tcl extension, 583–584
  Tcl modules, 239–241
  working with, 234–239
Packers
  defined, 298, 385
  geometry management, 396–404
Packing lists, 396
Packing space, 397
Padding options
  configuring gridder, 391
  configuring packer, 399
  defined, 341
  geometry management, 404–405
-padx option
  configuring gridder, 391
  configuring packer, 399
  event binding, 17
  geometry management, 404–405
  panedwindows, 336
  in widgets, 341
-pady option
  configuring gridder, 391
  configuring packer, 399
  event binding, 17
  geometry management, 404–405
  panedwindows, 336
  in widgets, 341
-palette option in photo images, 377–378
Panedwindows
  as geometry manager, 410
  overview, 334–337
Parameters, command, 576–578
Parcels
  defined, 362
  packing, 397
Parentheses, 81–82
Parsing
  defined, 22–24
  forcing with eval, 139–140
  operand substitution, 60–61
  string operands, 62
  strings, 543–544
  strings with scan, 76–78
  substitutions, 36
Pasting with clipboard, 473–474
Patch levels, 518
Patch releases, 717–718
PATH environment variable, 9, 200
Paths
  file pathtype command, 178–179
  module, 240
Pattern matching
  event patterns, 417–419
  Glob style, 78–79
  listing directory contents, 179–181
  with regular expressions, 79–87
  with switch, 133–136
Patterns
  event, 417–419
  option database entries, 507–508
peer text widget command, 464–466
Percent symbol
  operators and precedence, 55–56
  substitutions in scripts that handle
    events, 419–421
  in time and date strings, 254–256
Periods as separators, 7
photo images, 375–376, 377–383
Photos in window decoration, 485
pid command, 198, 209–210
Pipeline I/O, 203–204
pkg_mkIndex procedure, 231, 236–237
place command, 405
Placers
  defined, 385
  geometry management, 405
Platforms
  binding to named virtual events, 425–426
  building Tcl and Tk, 703–707
  file and directory name manipulation,
    176–179
  handling end-of-line conventions,
    188–190
  specific menus, 333

Platforms (*continued*)
  Starpacks, 245–246
  variable describing, 50
`plus` example, 144–145
Plus sign, 55–56
Pointer options in widgets, 341–342
`polygon` type, 434
Pop-up menus, 334
Ports, 210
Positional specifiers
  defined, 76
  using in source and translation strings, 95
Positioning points, 435
Positions, 482–483
POSIX
  errors, 222
  signal names, 222
Posted menus, 328–329
PostgreSQL, 728
PostScript, 445
`pow` function, 59
`power` example, 167
Precedence of operators, 55–58
Predefined variables, 49–50
Preferences, 84
Prefixes, 161–162
`pressed` state, 358
`PRIMARY` selection, 469
`printargs` example, 50
`printArray` example, 149
Printing array variables, 149
`printVars` example, 147
Priorities
  configuration option database, 509–510
  tag, 457
`proc` command
  creating procedures, 144–145
  defaults and variable numbers of arguments, 146–148
  defined, 12–13, 143
  Tcl code in namespace, 158–161
Procedures
  applying anonymous, 151–153
  basics, 144–145
  call by reference, 148–150
  command creation, 578–579
  command evaluation, 23
  commands, 143–144
  creating new control structures, 150–151
  declaring in namespaces, 158
  defaults and variable numbers of arguments, 146–148
  `factorial`, 12–14
  information about, 261–262
  libraries. *See* Libraries.
  local and global variables, 146
  Tcl as language, 14–15
Processes
  commands, 197–199
  configuring channel options, 204–205
  environment variables, 210
  event-driven channel interaction, 206–208
  IDs, 208–209
  invoking subprocesses, 199–203
  I/O to and from command pipeline, 203–204
  operating system utilities, 687–689
  reaping child processes, 689–690
  sending commands to Tcl programs, 214–217
  TCP/IP socket communication, 210–214
  termination, 199
Processing events, 423–424
Programming with POSIX Threads (Butenhof), 695
Progressbar widget, 349
Prompts, displaying command number, 287
Propagation
  geometry, 409
  using traces for, 51
Property, `RESOURCE_MANAGER`, 508–509
Protocols
  result, 569
  window management, 490–491
Pull-down menus, 329–331
`puts` command
  I/O to and from command pipeline, 203
  output buffering, 188
`pvar` procedure, 265
`pwd` command, 174, 179

Q

`Quadruple` modifier, 418
Qualified names
  inspecting namespace, 163–164
  manipulating, 161–162

Quantifiers, 84–85
Querying
  array elements, 46–47
  channel functions, 655
  with ifconfig, 585–592
  input focus, 494–495
Quoting
  with braces, 29–30
  with double quotes, 28–29
  in element names, 43
  expressions with braces, 61
  string manipulation, 62

R
\r in substitution, 27
radiobutton entry, 328
Radiobuttons, 315–316
raise command, 386, 407–408
rand function, 5, 59
Random Access, 190–191
Range
  extracting characters, 69
  regular expression atoms, 82–83
re1 | *re2* in regular expressions, 81
Read traces, 266–267
Readable files, 184
Readable time and date
  generating, 254–257
  scanning, 257–258
Reading
  files, 185–194
  Tcl variables, 555
readlink option, 185
readonly state, 358
Read-only variables, 266
Real operands
  defined, 54
  types and conversions, 63–64
Reaping, 689–690
rectangle type, 433
Redirection, I/O, 200–201
Redisplays, optimizing, 677–678
Redo
  history, 286
  text widget, 462
Reference
  call by, 148–150
  counting, 540–541
Reference documentation, xxxiii
regexp command, 66, 81
Registering
  background errors, 227–228
  channels, 653–654, 658–660
  file event handlers, 207–209
  new commands, 567–569
  toplevel windows as transient, 499
regsub command, 66, 87–88
Regular expressions
  matching, 622–623
  pattern matching with, 79–87
  using for substitutions, 87–88
Relational operators
  defined, 56–57
  expr syntax, 5–6
Relative paths, 178
Relative placement characters, 395–396
Relative qualified names, 161
Releases, 717–718
-relief option
  formatting text tags, 455
  for frames, 305
Removing variables, 49
Removing whitespace, 70
rename command, 252, 268
Renaming
  commands, 269
  files, 183
  tracing commands, 270
Repeating strings, 70
Repetition count, 259
Replacements, string, 72
Replacing
  elements from list objects, 644
  with text widget, 451–452
Repositories, installation, 722–724
Representing information, 526
Requested sizes of widgets, 387–388
Resizing windows, 481–482
Resolution, commands in namespaces, 170
Resolution paths, 170–172
Resource names, 524–525
RESOURCE_MANAGER property, 508–509
.result label, 16–17
Result protocols
  command creation, 569
  Tcl_AppendResult, 569–570
  Tcl_SetResult and interp->result, 570–572

Retrievals, selection, 469–470
`return` command
  basics of, 144–145
  defined, 144
  errors and exceptions, 220
  of `factorial`, 12–13
  generating exceptions, 224–226
  return options dictionary, 604–605
Return options dictionary, 225–226
Returns
  managing return options dictionary, 604–605
  normal and exceptional, 35
Right-shifting, 57
`-rmargin2` option in text tags, 455
Romeo and Juliet (Shakespeare), 469
Root items, 351
Root windows, 292, 296
`ROOT.msg`, 93
ROT13 channels, 663–669
`round` function, 59, 64
Rows
  gridder options, 389–391
  spanning gridder, 393–394
`ruler` example, 435–436
Runtime information, 264

S
Safe interpreters, 277–279
Safety, thread, 697
Saving sessions, 490
Scalar variables, 39
Scales, 321–323
`scan` command
  defined, 54, 66
  parsing strings with, 76–78
Scoping for variables, 143
Screens
  distances, 307
  Tk, 296
Script files
  `info script`, 263
  overview, 8–10
Script libraries. *See* Libraries.
Scripting languages
  defined, xxxi
  Tcl/Tk benefits, xxxii–xxxiii
Scripts
  dynamically building, 548
  executing in namespaces, 158–161
  generating errors from, 222
  multithreaded Tcl, 727
  packaging as Starkits, 242–246
  substitutions in, 419–421
  supplying selection with, 471–473
  syntax overview, 21–22
  Tk, 297
Scroll regions, 444–445
Scrollbars
  commands for interconnection, 300
  overview, 318–321
Scrolling
  canvas widget, 444–445
  themed widgets, 364–365
`sdx` Starkit, 243–245
Searching
  with `array names`, 47
  canvas specifications, 438–439
  glob-style pattern matching, 78–79
  hash tables, 636–637
  lists, 110
  with message catalogs, 91–95
  with regular expressions, 79–87
  simple string, 70–71
  with text widget, 451–452
Security and `send`, 217
`sel` tag, 453–454
Selected checkbuttons, 313
`selected` state, 358
Selection
  `clipboard`, 473–474
  commands, 468–469
  DND, 474–475
  locating and clearing, 470–471
  managing treeview, 355–356
  overview, 467–468
  retrievals, types and, 469–470
  supplying with Tcl scripts, 471–473
`selection clear` command, 468, 471
`selection` command, 301
`selection get` command, 468, 469–470
`selection handle` command, 468
`selection own` command, 468, 470
Selection owner model, 467

`Selection` virtual event, 458
Selectors, checkbutton, 313
Semicolons, 5
`send` command, 214–217
`separator` entry, 328
Separator widget, 350
Separators
  in basic syntax, 21–22
  defined, 7
  disabling, 28
  variable substitution, 44
Sequences, event, 419
Server sockets, 212–214
Service-oriented architecture (SOA), 728
Session management, 490
`set` command
  evaluating, 23
  simple variables and, 41
  working with variables, 40
Setting
  command parameters, 576–578
  Tcl variable values, 553–555
  traces, 558–559
Shapes, canvas, 433–436
Shared libraries, 232
Shared objects, 541–542
Sharing
  channels between interpreters, 279–280
  between peer widgets, 464–466
Shells, 596–597
Shifting operators, 57
Shimmering, 42
Shortcuts
  binding to named virtual events, 425–426
  event pattern, 418–419
  implemented by `unknown`, 286–287
  menu, 332–333
  notebook, 349
`-show` option for entry widgets, 325
`-side` option for packer, 399–400
Signals, 690–694
`sin` function, 59
Single-file distribution, 242–246
`sinh` function, 59
Size
  channel buffering mode, 205
  common widget options, 339–340
  dictionary object, 647
  examining dictionary, 122
  file information options, 184–185
  window management, 481–482
`-size` option in fonts, 370–371
Sizegrip widget, 350
Sjölander, Kåre, 726
`-slant` option in fonts, 370
Slave interpreters
  creating, 531–532
  defined, 274–281
Slave widgets
  defined, 320–321
  geometry management, 387–389
Sliders
  in scrollbars, 318
  scrolling single widget, 319–320
Snack, 725–726
Snit, 726
SOA (service-oriented architecture), 728
`socket` command, 198, 210–212
Socket communication, 210–214
`socket -server` command, 198
Sorting
  dictionaries, 123
  lists, 111
Sound support, 725–726
Source code
  building Tcl and Tk, 703–704
  compiling Tcl/Tk from, 719
`source` command
  autoloading and, 233
  defined, 66, 132
  executing from files, 140–141
  safe interpreters, 278–279
  system encodings, 90
SourceForge, 703, 719
Space characters
  `append` command and, 48
  backslash substitution, 27
  multidimensional arrays, 46
`-spacing` options in text tags, 456
Special characters
  backslash substitution, 26–27
  creating commands as lists, 113–114
  glob-style pattern matching, 79
  quoting with braces, 29–30
  in regular expressions, 80–83
  variables and substitutions, 11

Specifiers, conversion
  generating strings with format, 74–76
  using in source and translation strings, 94–95
Specifiers, event, 285
Specifiers, positional, 76
Spencer, Henry, 79
Spinboxes, 324–325
Spinning, 324
Split characters, 112
split command
  converting between lists and strings, 111–113
  defined, 103
sprintf procedure, 74–75
sqlite2, 728
sqrt function, 59
Square brackets
  command substitution, 26
  invoking substitution, 11
srand function, 59
sscanf procedure, 76
Stack traces, 605–608
Stacked channels, 662–663
Stacking order
  canvas item, 435
  widget, 407–408
Standard error, redirection, 201
Standard input
  command completeness, 627–628
  redirection in exec, 201
Standard output
  channel I/O, 661
  redirection in exec, 201
Standard Tcl Library, 725
Star-crossed lovers, 470
Starkits
  defined, 229
  packaging scripts as, 242–246
Starpacks, 245–246
stat option, 184–185
state widget command, 359
States
  event, 416
  representing with checkbuttons, 314
  themed widget, 358–360
  widget, 339
  window management, 484–485
Static variables, 146
Statistics, 638
stderr channel
  defined, 187
  manipulating in C, 660–661
stdin channel
  defined, 187
  manipulating in C, 660–661
stdout channel
  defined, 187
  manipulating in C, 660–661
-sticky option
  configuring gridder, 391, 392–393
  panedwindows, 336
Storage
  with clipboard, 473–474
  internal data, 41–42
  Tcl module, 239
Stretch behavior of gridder, 394–395
-stretch option for panedwindows, 336
string bytelength command, 67
string command, 23
string compare command, 67, 71
string equal command, 67, 71
string first command, 67, 70–71
string index command, 67, 69
string is command, 67, 73–74
string last command, 67, 71
string length command, 67, 70
String manipulation
  binary strings, 95–99
  character set issues, 89–91
  commands, 65–69
  comparisons, 71
  extracting characters, 69
  generating with format, 74–76
  glob-style pattern matching, 78–79
  length, case conversion, trimming, and repeating, 70
  message catalogs, 91–95
  parsing with scan, 76–78
  pattern matching with regular expressions, 79–87
  replacements, 72
  simple searching, 70–71
  substitutions with regular expressions, 87–88
  types, 73–74

string map command, 67–68, 72
string match command, 68, 78–79
string range command, 68, 69
string repeat command, 68, 70
string replace command, 68, 72
string tolower command, 68
string totitle command, 68
string toupper command, 68, 70
string trim command, 68, 70
string trimleft command, 68, 70
string trimright command, 68, 70
STRING type, 469–470
String utilities, C language
  command completeness, 627–628
  dynamic, 617–621
  functions, 611–617
  handling Unicode and UTF-8, 625–627
  regular expression matching, 622–623
  string matching, 621–622
  working with character encodings, 624–625
string wordend command, 69
string wordstart command, 69
Strings
  accompanying exceptions, 224–225
  command evaluation, 22–24
  constructing with append, 49
  converting between lists and, 111–112
  hash table keys, 632
  internal storage of data, 41–42
  manipulation commands, 19
  manipulation expressions, 62–63
  object, 537
  parsing, 543–544
  representing values, 4–5
  time and date, 254–258
  variables and substitutions, 11
Stubs, 584–585
Styles
  defined, 343
  themed widget, 360–364
Subcommands
  image, 378–379
  working with ensembles, 164–169
Subexpressions
  defined, 81–82
  non-capturing, 85
Subpatterns, 82
Subprocesses, 199–203
Substitutions
  backslash, 26–27
  call by reference, 148–149
  command, 26
  expression, 60–61
  in file names, 176
  inside braces, 12–13, 29–30
  inside double quotes, 28–29
  overview, 11
  with regular expressions, 87–88
  in scripts, 419–421
  with switch, 134
  syntax overview, 35–37
  Tcl as language, 14–15
  variable, 24–25, 43–45
Substrings, 86
Subtraction operator, 5–7, 55–56
sum example, 148
Support
  characters classes supported by string is, 73–74
  exec, 200
  operators and precedence, 55–58
  regular expression, 80
  Tcl/Tk benefits, xxxii
  for virtual file systems, 194
Suspended processes, 202, 672–673
SWIG, 729
switch command, 132, 133–136
Sybtcl, 728
Symbolic links, 185
Synchronized scrolling, 320–321
Synonyms, 308
Syntax
  argument expansion, 30–32
  backslash substitution, 26–27
  command evaluation, 22–24
  command substitution, 26
  comments, 32–35
  defined, xxxiv
  normal and exceptional returns, 35
  option database entries, 507–508
  quoting with braces, 29–30
  quoting with double quotes, 28–29
  regular expressions, 80
  scripts, commands, and words, 21–22
  substitutions, 35–37
  use of braces, 12–13
  variable substitution, 24–25

System call errors, 196
System encodings, 89–90

T

`\t` in substitution, 27
Tab identifiers, 347
`-tab` option in text tags, 456
Tables, hash. *See* Hash tables.
Tabs
  formatting text tags, 456
  in notebook widgets, 347–348
`-tabstyle` option in text tags, 456
Tags
  canvas widget, 433
  event-binding hierarchy, 422–423
  manipulating items with canvas widget, 436–439
  text widget, 447, 453–458
  treeview item, 256–257
`tan` function, 59
`tanh` function, 59
Targets, selection, 469–470
`TARGETS` selection target, 470
`.tcl`, 9
Tcl (tool command language)
  additional features, 19
  benefits of, xxxii–xxxiii
  building, 703–707
  building threaded, 697
  control structures, 12–14
  creating commands. *See* Command creation.
  defined, xxxi
  embedding. *See* Embedding Tcl.
  evaluating code. *See* Evaluating Tcl code.
  evaluating code in namespaces, 158–161
  event binding, 15–18
  generating errors from scripts, 222
  getting started, 3–6
  "Hello, World!" with Tk, 6–8
  installing, 717–719
  interpreters. *See* Interpreters.
  as language, 14–15
  script files, 8–10
  script libraries. *See* Libraries.
  sending commands to Tcl programs, 214–217
  supplying selection with scripts, 471–473
  variable access. *See* Accessing Tcl variables.
  variables and substitutions, 11
Tcl and C integration
  action-oriented vs. object-oriented, 525–526
  overview, 521–523
  representing information, 526
  resource names, 524–525
  Tcl vs. C, 523–524
Tcl Extension Architecture (TEA), 707–714
Tcl internals
  commands, 247–253
  `info` command, 259–264
  renaming and deleting commands, 269
  slave interpreters, 274–281
  time and date manipulation, 254–259
  time delays, 253–254
  timing command execution, 259
  tracing array variables, 268–269
  tracing commands, 270–272
  tracing operations on simple variables, 264–267
  unknown commands, 272–274
Tcl objects
  byte arrays, 540
  composite, 540
  dictionaries, 644–648
  dynamic nature of, 539
  fetching C values from, 538–539
  functions, 534–537
  gizmos, 629–630
  image, 375
  lists, 642–644
  memory allocation, 544
  namespaces and, 171–172
  new types, 542–543
  numerical, 537–538
  overview, 533–534
  parsing strings, 543–544
  reference counting, 540–541
  shared, 541–542
  string, 537
Tcl resources
  books, 732–733
  online, 731–732
`TCL_ALL_EVENTS` symbol, 679–680
`Tcl_AppendResult` procedure, 569–570

Tcl_CancelIdleCall procedure, 677
Tcl_CreateHashEntry procedure, 431, 433–434
Tcl_CreateSlave function, 531–532
Tcl_CreateTimerHandler procedure, 672, 676–677
Tcl_DeleteHashEntry procedure, 637
Tcl_DeleteHashTable procedure, 633, 637
Tcl_DeleteInterp function, 531
Tcl_DeleteTimerHandler procedure, 676–677
Tcl_DetachPids procedure, 685, 690
TCL_DONT_WAIT flag, 680
Tcl_DoOneEvent procedure, 677–680
Tcl_DoWhenIdle procedure, 677
TCL_EVAL functions, 545–547
TCL_EXCEPTION flag, 673
TCL_FILE_EVENTS flag, 680
Tcl_FileProc procedure prototype, 671–673
Tcl_FindHashEntry procedure, 635
Tcl_FirstHashEntry procedure, 636
Tcl_GetHashKey macro, 636
Tcl_HashSearch type, 636
Tcl_HashStats procedure, 638
Tcl_HashTable type, 632–633
TCL_IDLE_EVENTS flag, 680
Tcl_IdleProc procedure prototype, 677
Tcl_InitHashTable procedure, 632–633
Tcl_Interp function, 527–528, 529
Tcl_NextHashEntry procedure, 636–637
TCL_OBJ function
  defined, 534–535
  regular expression matching, 623
TCL_ONE_WORD_KEYS symbol, 632
Tcl_Panic function, 608–609
tcl_platform array, 50
TCL_READABLE flag, 655–656, 662–663, 673–675
Tcl_ReapDetachedProcs procedure, 685, 690
Tcl_SetHashValue macro, 633–634
Tcl_SetResult procedure, 570–572
Tcl_SignalId procedure, 693
Tcl_SignalMsg procedure, 693
TCL_STRING_KEYS symbol, 630, 632
TCL_TIMER_EVENTS flag, 680
Tcl_TimerProc procedure prototype, 672, 676
Tcl_TimerToken type, 676
TCL_WRITABLE flag, 655–656, 662–663, 673–675
Tcl/Java Project, 728–729
Tclkits
  defined, 229
  installing, 719
  packaging scripts as Starkits, 242–243
Tcllib, 725
TclODBC, 728
tclsh application
  creating new shells, 596–597
  getting started, 3–6
  script execution, 10
  variables and substitutions, 11
TclSOAP, 728
tclvfs extension, 194–196
TclX (Extended Tcl), 730
TclXML, 727
TCP/IP socket communication, 210–214
tDOM, 727
TEA (Tcl Extension Architecture), 707–714
teacup, 722–724
TEApot, 722–724
Tear-off menus, 329
Termination
  process, 199
  thread, 698
Ternary operator, 56, 58
text command, 447
Text editor example, 428–431
Text files, 185–194
-text option
  button command, 7–8
  for labels, 309–310
  themed widgets, 365
text type, 434
Text widget
  basics, 447–450
  embedded images, 460–462
  embedded windows, 459–460
  gridded windows, 482–483
  indices and marks, 450–451
  peer, 464–466
  search and replace, 451–452
  tags, 453–458
  -undo, 462–463
  virtual events, 458

`-textvariable` option
  event binding, 16
  for labels, 309–310
  themed widgets, 365
`tgrep` script, 186–188
Themed widgets
  basics, 305
  vs. classic widgets, 343–345
  combobox, 345–346
  defined, 303–304
  notebook, 346–349
  other standard options, 364–365
  progressbar, 349
  separators, 350
  sizegrip, 350
  states, 358–360
  styles, 360–364
  treeview, 350–357
Themes
  defined, 343
  widget styles, 360–361
Threads
  building threaded Tcl, 697
  condition variables, 699–701
  creating, 697–698
  extension, 727
  functions, 695–697
  interacting with blocking channels, 206
  miscellaneous, 701–702
  multiple interpreters, 275
  mutexes, 698–699
  safety, 697
  terminating, 698
Tilde
  in file names, 176
  precedence and, 55–56
Time
  command execution, 259
  delay commands, 253–254
  event processing, 423–424
  manipulation, 254–259
`time` command, 252, 259
Time zones, 256
Timer events, 676–677
`-titlepath` attribute, 488
Titles, window, 485
Tk
  applications, toplevel widgets, and screens, 296
  benefits of, xxxii–xxxiii
  building, 703–707
  commands for fonts and images, 367–369
  commands for interconnection, 300–301
  configuration options. *See* Configuration options.
  creating and destroying widgets, 297–298
  defined, xxxi
  `focus` commands, 492
  fonts, 369–375
  geometry managers, 298–299. *See also* Geometry managers.
  "Hello, World!" example, 6–8
  images, 375–383
  installing, 717–719
  introduction to windowing systems, 292–293
  odds and ends, 513–518
  overview, 291
  scripts and events, 297
  `send` command, 214–217
  `tcl_platform` array, 50
  widget commands, 299–300
  widgets, 294–295. *See also* widgets.
`tk appname` command, 513, 517
`tk` command, 516–517
`tk inactive` command, 513, 516–517
`tk scaling` command, 513
`tk windowingsystem` command, 513, 516
`TK_APPLICATION` selection target, 470–471
`Tk_DoOneEvent` procedure, 679
`tk_focusFollowsMouse` command, 492, 494
`TK_LIBRARY` environment variable, 517–518
`tk_library` variable, 517–518
`Tk_MainLoop` procedure, 678–679
`tk_patchlevel` variable, 518
`tk_version` variable, 518
`TK_WINDOW` type, 471
TkCon, 724
`tk::Priv`, 518
`tkwait` command
  defined, 492–493
  invoking event loops, 423–424
  modal interactions, 497–499
Tokens, 578
Tool command language (Tcl). *See* Tcl (tool command language).
`-toolwindow` attribute, 488
Toplevel widgets, 296, 308–309

Toplevel windows
  defined, 292, 296
  managing position, 482–483
  sizing, 481–482
`-topmost` attribute, 487
`trace add command` command, 252, 270–271
`trace add execution` command, 252, 271
`trace add variable` command, 252, 267
`trace` command
  array variables, 268–269
  commands, 270–272
  defined, 50–51
  operations on simple variables, 264–267
`trace info command` command, 252
`trace info execution` command, 252, 272
`trace info variable` command, 252, 267
`trace remove command` command, 252, 272
`trace remove execution` command, 252, 272
`trace remove variable` command, 252, 267
Tracing
  adding to stack trace in `errorInfo`, 605–608
  array variables, 268–269
  callbacks, 559–561
  command creation, 579
  commands, 270–272
  multiple, 561
  operations on simple variables, 264–267
  setting and unsetting traces, 558–559
  whole-array, 561
Transferring channels between interpreters, 279–280
Transformation formats, 89–91
Transients
  custom dialogs, 499–500
  defined, 486
Translation
  binary to text, 190–191
  creating localized message files, 93–94
  using in conversion specifiers in, 94–95
`-transparentcolor` attribute, 488
Treeview widget, 350–357
Trimming strings, 70
`Triple` modifier, 418
Tristate values
  defined, 314
  of radiobuttons, 315–316
Troughs, 318
True values, 74
Types
  adding selection to widgets, 471–473
  canvas widget, 433–436
  channel definition functions, 655–656
  conversions and, 63–64
  creating new channel, 661–669
  event, 414–415
  fetching C values from Tcl objects, 538–539
  finding file, 180
  identifying file, 183
  new object, 542–543
  numerical object, 537–538
  selection and, 469–470
  string, 73–74
  variable linking, 557

U

`\u`*`hhhh`* in substitution, 27
`-underline` option
  configuring font, 371
  formatting text tags, 455
`-undo` in text widget, 462–463
Undocking window, 488–489
Unicode
  character set issues, 89
  characters supported by `string is`, 73–74
  handling strings, 625–627
  Tcl backslash substitution, 26–27
  Tcl support, xxxii
  working with character encodings, 624–625
`-uniform` option
  configuring gridder, 392
  stretch behavior and, 394–395
Unix
  building Tcl and Tk on, 704–705
  bundled Tcl distributions, 718
  executable scripts on, 9
  `tcl_platform` array, 50
`unknown` command
  defined, 252–253
  managing, 272–274
  shortcuts implemented by, 286–287
`-unknown` option in ensembles, 167–169

`unknown` procedures, 233
`Unmap` event, 415
Unposted menus, 328–329
`unset` command
  variable removal, 49
  working with variables, 40
Unsetting
  Tcl variables, 556
  traces, 558–559
`update` command
  defined, 513
  invoking event loops, 423–424
  updating widgets, 514–516
Updating
  dictionaries, 120–121
  dictionary values, 123–126
  installation repositories, 723
Uplevel addressing, 144
`uplevel` command, 144
`upvar` command
  call by reference, 148–150
  defined, 51, 144
  `env` variable and, 268–269
User interface creation with Tk. *See* Tk.
UTF-8
  character set issues, 89–91
  handling strings, 625–627
  working with character encodings, 624–625
Utilities, operating system. *See* Operating system utilities.
Utilities, string. *See* String utilities.

V
`\v` in substitution, 27
Validation for entry widgets, 325–327
Values
  basic dictionary structure, 118–120
  changing with `incr` and `append`, 47–49
  creating commands as lists, 113–114
  dictionary objects, 644–648
  examining dictionaries, 122–123
  in expressions. *See* Expressions.
  fetching C from Tcl objects, 538–539
  generating strings with `format`, 74–76
  in hash tables, 629, 631–632
  option database entries, 507–508
  setting Tcl variable, 553–555
  simple variable, 41
  storage as strings, 41–42
  string representation, 4–5
  updating dictionary, 123–126
  variables and substitutions, 11
`variable` command, 159–160
Variable linking, 556–558
Variable substitution
  defined, 11
  overview, 43–45
  syntax overview, 24–25
Variables
  accessing from C. *See* Accessing Tcl variables.
  accessing from other namespaces, 169–170
  accessing with `upvar`, 148–150
  arrays, 42–43
  `auto_path`, 234
  command evaluation, 23–24
  commands, 39–40
  declaring in namespaces, 159
  environment, 210
  `errorCode`, 221–222
  global, 13–14
  `incr` and `append` commands, 47–49
  information about, 260–261
  inspecting namespaces, 163–164
  listbox, 317
  local and global, 146
  managed by Tk, 517–518
  multidimensional arrays, 45–46
  overview, 11
  predefined, 49–50
  preview of other facilities, 50–51
  querying array elements, 46–47
  radiobutton options, 315
  setting `errorCode`, 603–604
  simple and `set`, 41
  slave interpreter, 274
  Tcl's internal storage of data, 41–42
  `-textvariable` option, 309–310
  thread condition, 699–701
  tracing array, 268–269
  tracing operations on simple, 264–267
  `unset` and `array unset`, 49
  waiting and, 499
Vasiljevic, Zorand, 727

Versions
  installing Tcl and Tk, 717–718
  Tcl interpreter, 264
  `tclsh` application, 3–4
  Tk variables, 518
  using packages, 235
  utility package commands, 238
Virtual events
  binding to named, 425–426
  generating, 427–428
  text widget, 458
Virtual file systems
  file access, 194–196
  interaction, 683
  Starkits, 242
Virtual root window managers, 483
Volumes, 179
`vwait` command
  defined, 199
  entering Tcl event loop with, 206–207
  invoking event loops, 423–424

W

Waiting
  condition variables for threads, 699–701
  `tkwait` command, 497–499
`waitWindows` example, 499
`-weight` option
  configuring font, 371
  configuring gridder, 392
  stretch behavior and, 394–395
Welch, Brent, 727
`while` command
  defined, 132
  of `factorial`, 12–13
  looping, 136–138
Whitespace characters
  backslash substitution, 27
  in basic syntax, 22
  defined, 4
  variable substitution, 44
Whole-array traces, 561
`wide` function, 59, 64
Wide types, 538
Widget commands
  basics, 304–305
  `cget`, 511–512
  configuration options, 505–506
  `configure`, 511–512
  defined, 299–300
  evaluating, 23
  event binding, 18
`widget delete` command, 448
`widget get` command, 448
`widget insert` command, 448
`widget replace` command, 448
Widgets
  additional features, 19
  basics of, 304–305
  buttons, 312–315
  canvas. *See* Canvas widget.
  color options, 307–308
  commands for interconnection, 300–301
  creating and destroying, 297–298
  defined, 292
  entries, 323–327
  event bindings, 15–18
  events. *See* Events.
  frames, 306–307
  geometry management, 409–411
  geometry managers, 298–299
  "Hello, World!" example, 7–8
  joinable thread support, 698
  labelframes, 312
  labels, 309–312
  listboxes, 317–318
  menus, 327–334
  odds and ends, 513–518
  other common options, 339–342
  overview, 294–295, 303–304
  panedwindows, 334–337
  scales, 321–323
  scrollbars, 318–321
  selection. *See* Selection.
  stacking order, 407–408
  standard dialogs, 337–339
  text. *See* Text widget.
  vs. themed widgets, 343–345
  toplevel, 296
  toplevels, 308–309
`-width` option
  configuring photo images, 378
  panedwindows, 337
  in widgets, 339–340

Wildcards
  in glob-style pattern matching, 180
  in patterns, 507
Window manager protocols, 490–491
Window managers
  commands, 478–481
  decorations, 485–486
  defined, 293, 298
  dockable windows, 488–489
  gridded windows, 483–484
  overview, 477–478
  session management, 490
  special handling, 486–487
  states, 484–485
  system-specific window attributes, 487–488
  window close, 489–490
  window positions, 482–483
  window sizes, 481–482
`window` type, 434
Windowing commands
  additional features, 19
  "Hello, World!" with Tk, 6–8
Windowing systems
  `clipboard` command, 471–473
  introduction to, 292–293
  stacking order, 435
  `tk` command, 516–517
Windows
  binding to named virtual events, 425–426
  building Tcl and Tk on, 707
  building TEA extension on, 714
  cursor options, 342
  defined, 292
  embedded in text widget, 459–460
  `exec` support, 200
  executable scripts on, 9–10
  input focus, 493–495
  invoking `wish` on, 6
  modal, 495–499
  platform-specific menus, 333
  system-specific window attributes, 487–488
  `tcl_platform` array, 50
  waiting and, 498
`winfo` command, 513, 516
`wish` interpreter
  executing script files with, 9–10
  "Hello, World!" example, 6–8
  variables and substitutions, 11
  versions, 4
Withdrawn window state, 483–484
`wm aspect` command, 482
`wm attribute` command, 487–488
`wm client` command, 490
`wm` command
  defined, 301, 477–478
  geometry management, 410
  list of subcommands, 478–480
`wm command` command, 490
`wm deiconify` command
  creating custom dialogs, 500, 504
  defined, 485
`wm forget` command, 488–489
`wm geometry` command
  window positions, 482–483
  window sizes, 481
`wm group` command, 486
`wm iconbitmap` command, 486
`wm iconify` command, 485
`wm iconmask` command, 486
`wm iconname` command, 485–486
`wm iconphoto` command, 485–486
`wm iconposition` command, 486
`wm iconwindow` command, 486
`wm manage` command, 488–489
`wm override-redirect` command, 487
`wm protocol` command, 490–491
`wm state` command, 485
`wm title` command, 485
`wm transient` command, 486
`wm withdraw` command
  creating custom dialogs, 500, 504
  defined, 485
`WM_DELETE_WINDOW` protocol, 490–491
Words
  command evaluation, 22–24
  defined, 4
  substitution in. *See* Substitutions.
  syntax overview, 21–22
`-wrap` option in text tags, 456
Writable files, 184
Write traces, 266–267
Writing
  files, 185–194
  Tcl variable values, 553–555

X
`\x`
  glob-style pattern matching, 79
  in regular expressions, 80
`x` in gridder, 396
X Window System
  selection, 467
  `send`, 215
X11
  platform-specific menus, 333
  system-specific window attributes, 488
`.Xdefaults` file configuration, 508–509
`\x`*hh* in substitution, 27
XML programming, 727–728
XOTcl, 726

Y
`-yscrollcommand` option, 319–320

Z
Zdun, Uwe, 726
Zombies, 689–690
Zoomed state for windows, 488