command shell
by Tony Lewis
tlewis@exelana.com

The Perl script provides a command shell to which you can add your own commands. To create your custom shell, extract the cs.tgz file and run customize.pl. (See the *Customization* section beginning on page 38 for instructions.) After customization you will have a functional command shell, with several built-in commands and one predefined command (that command is 'test', which you may choose to delete).

The latest version is available at http://www.exelana.com/techie/perl/cs.html

Creating your command shell

The customization script will create three files (using a prefix that you specify in Customization step 9):

File	Description
xx.pl	This is the file that the user will run from the command line.
xx_defs.pl	Loaded by xx.pl; this file contains the definitions for your command shell.
xx_impl.pl	Loaded by xx.pl; this file contains the implementation of your command shell.

In theory you could combine all this code into a single file, but I recommend that you keep the definitions and the implementation separate. If your implementation contains a lot of commands, you might want to break xx_impl.pl into multiple files.

Note that xx.pl is a wrapper that loads cs_main.pl. This allows you to replace the command shell code with a new version (although you should consult the README file about any compatibility issues).

This documentation corresponds to version 3.1 released on September 5, 2017.

Built-in Commands

The command shell contains several commands to support users of your command shell as well as commands to support you as the script developer.

User Commands

The following built-in commands are available to users of your command shell.

Command	Description
about	Display information about the command shell.
cd *path*	Change the working directory.
exit	Exit perl, system, or another added mode and return to the main mode
help *[topic]*	Display information about available commands
perl	Enter “perl” mode to evaluate Perl expressions
pwd	Print the current working directory.
quit	Exit the command shell
system	Enter “system” mode to enter operating system commands.

Shell Developer Commands

The following built-in commands are available to shell developers.

Command	Description
debug *[level]*	Change the debugging level (options: off on 0 1 2 3) Also "debug *flag [level]*"
dump *[topic]*	Dump internal information structures; see “help dump”
edit *file*	Edit a file (Note: "defs" or "impl" will open your xx_defs.pl or xx_impl.pl file.
editlib *lib*	Find a system library file.
findlib *lib*	Find a system library; for example “findlib Term::ReadKey”
reload	Reload the script (typically after editing one of the source files)
storage *file*	Examine a file created by storeData

Signal Processing

The command shell defines signal handlers for the following signals:

Signal	Description
CHLD	If requested, reloads command shell in a new child process; otherwise exits program.
INT	Catches ^C interrupts from the keyboard and sets the variable $int to 1. If your commands take more than a few seconds to complete, you should monitor $int and gracefully exit the command. If the user presses ^C five times without the command responding, the user is presented with the option of reloading the program.
__WARN__	Adds stack dump to warning messages.
WINCH	Catches changes to the size of the terminal and updates internal variables.

Debugger

The command shell defines a debugger that can be invoked inline allowing you to examine or change variables and to execute subroutines.

To call the debugger, insert one of these statements:

```
return if debugger(sub { return eval($_[0]) });
map{next if m/next/;last if m/last/;return if m/return/}
   debugger(sub{return eval($_[0])});
```

The first statement will return from the calling subroutine if you enter "return". The second statement also handles next and last so that you can manipulate a loop.

Commands available within the debugger:

Command	Description
exit	Exit the debugger and resume normal processing
last	Exit the debugger and instruct caller to process a last statement
next	Exit the debugger and instruct caller to process a next statement
return	Exit the debugger and instruct caller to process a return statement
skip [n]	Skip the next *n* invocations of the debugger
stack	Print the call stack

Note: The main loop of the command shell will reset the skip count to zero at the start of a command.

The debugger can be invoked during any call to getReply (including calls to getNumber, getNumberList, and yesNo) by entering ":debugger" as the response.

Initializing and Configuring the Command Shell

Before the first command can be processed, the command shell must be initialized and configured. This processing is setup in the xx_defs.pl source file by the customization script (see *Customization* on page 38), but you can modify and extend the configuration.

initHelp

The first initialization call that your code must make is to initHelp, which is described on page **Error! Bookmark not defined.**. This call is created by the Customization script (steps 1 through 4); see page 38.

addAbout

Add a description of your customized command shell to the about command using addAbout, which is described on page 17. If this routine is called multiple times, the output is concatenated on subsequent lines of the output of the about command.

This call is created by the Customization script (steps 10 through 13); see page 38.

addHelpGroup

Add a new help group to the command shell using addHelpGroup, which is described on page 19. This call is created by the Customization script (step 5); see page 38.

addHelp

You can add additional help text to a help group using addHelp, which is described on page 19. Note that help lines are displayed in the order they are added. If you want the help text to begin with an explanation followed by the commands in the group, add the explanation before calling addCommand.

setHistory

Set the file path for storage of command history, which is described on page 32. This call is created by the Customization script (step 6); see page 38.

Modes of Operation

The command shell supports several modes of operation. When the program is running in a specific mode, the default action is for all commands to be processed in that mode. To execute a single command in another mode, the user will add the mode prefix to the command.

Three modes are implemented by the main script:

Mode	Prefix	Description
main		This mode of operation to which you will add your own commands (set in Customization steps 7 and 8)
system	!	Executes operating system commands and displays the output
perl	p!	Evaluates Perl expressions

Four additional optional modes are implemented by support routines (described below):

Mode	Prefix	Description
cgi	c!	Execute commands on a remote server (see *CGI Mode* below)
db	d!	Query a MySQL database (see *Database Support* below)
network	n!	Process network commands (see *Network Support* below)
xml	x!	Explore XML content (see *XML Support* below)

For example, from the main mode:

Mode	Description
test	will execute your 'test' command.
p!hex("4d")	calls the Perl hex function and prints the result: 77
perl	will switch into perl mode where Perl commands can be entered without the p! prefix.
exit	will return to the main mode from perl mode.

addMode

To create additional modes, call the *addMode* function, which is described on page 19.

Adding Commands and Aliases

The user interacts with the program by entering commands. There are predefined commands for each mode name, each mode prefix, and several internal commands (such as debug and help).

You can add commands to your custom shell that will be executed in any mode that you have defined. The command shell will process user input, detect your command, and call a subroutine that you define with any arguments that the user supplies. If the command cannot be parsed, the *error routine* for the mode will be called with the user's input.

Any command can have one or more aliases, which are created by calling addAlias.

You can specify positional arguments and switches for the command. Positional arguments must appear in the order defined although switches may be interspersed with the positional arguments.

addCommand

To add a command, call the addCommand function, which is described on page 17.

addAlias

To add an alias for a command, call the addAlias function, which is described on page 17.

setOptionalUndef

To change the default behavior of optional arguments, call setOptionalUndef, which is describe on page 33.

Implementing Commands

You will implement your commands in the xx_impl.pl file, which initially contains the following routines that you may use to customize the behavior of the command shell:

Routine	Description
localInit	The command shell calls this routine before the user's first command; put any initialization code here. This is a good place to call loadData or loadNamedData to restore any persistent data. If you plan to define any debug flags, you should call addDebugFlag from localInit.
localExit	The command shell calls this routine before the program is exited or restarted; put any initialization code here. This is a good place to call storeData or storeNamedData to save any persistent data.
localEditor(path,editor)	The command shell calls this routine before choosing an editor. This routine can change $$path to point to a new executable and $$editor to be the name of the chosen executable. Note that the user can configure the editor by setting the environment variable EDITOR.
localFileToEdit(path)	The command shell calls this routine before editing a file (in an external editor). The choice of editor is configured by the user. This routine can change $$path to point to a new location. For example, it might look for the file by name in several folders and point $$path to the path of the file in one of those folders.
localPreInput(input)	The command shell calls this routine before processing user input. You can perform internationalization, alias substitution, etc. here. Update $$input with the string that the shell should process. Set $$input to the null string to prevent the main loop from taking further action on it.
localPostInput(input)	The command shell calls this routine after processing user input. You can override the quit and reload commands by changing the value of $quit to 0. Obviously, you should only do this if the user concurs; for example: `$quit = yesNo("You changed your input file without saving. Are you sure you want to quit? ");`

Commands

For each command that you have added via addCommand, you must provide a subroutine to implement that command. Note that the subroutine will receive zero or more arguments based on the command syntax provided to addCommand, which is described on page 17.

Debugging

Note that while it is generally good form to hide your variables using "my", if you leave them exposed, you can view and manipulate them using the built-in Perl mode.

CGI Mode

To include support for CGI requests to your command shell, add this line to xx_impl.pl:

```
require "cs_cgi.pl";
```

You must call *cgiInit* from your *localInit* routine and call *cgiExit* from your *localExit* routine.

CGI mode and the c! prefix allow you to execute system commands in the same environment as a remote system executes CGI scripts. It is derived from cgi-shell by Michael Pradel.

To use CGI mode, the three files in the server directory must be installed on the remote server in a directory where CGI commands can be executed. The files must have execute permission. Enable CGI mode in the command shell with a call to addCGI, which is described on page 17.

The addCGI routine can be called multiple times for different sites.

User Commands

This file will add *cgi* mode with the following built-in commands available to users of your command shell.

Command	Description
get	get file [dest] [--replace] [--gzip] Get <file> from the remote site into <dest> (or current local working directory) --replace will replace <dest> file if it exists --gzip will cause the file to be ZIP'ped before transfer
log	Show the CGI log.
put	put file [dest] [--replace] [--gzip] Put <file> to the remote site into <dest> (or current remote working directory) --replace will replace <dest> file if it exists --gzip will cause the file to be ZIP'ped before transfer
select	select [site] Select the current <site> (or display the current site if none); display all if '*'

All other commands are passed to the server for processing.

Security

The CGI interface uses basic authentication. The file _htaccess in the server directory contains the following example Apache configuration directives to require a user name and password to access the server:

```
<FilesMatch "^(cgi-shell-server|getFile|putFile)\.cgi$">
  AuthType Basic
  AuthName "CGI-Shell"
  AuthUserFile "/home/public_html/cgi-bin/.cgishell"
  require valid-user
</FilesMatch>
```

You can copy these directives into an existing .htaccess file or simply rename the _htaccess file that is provided and then change the path for the AuthUserFile to match your server directory structure.

Support Routines

This file defines the following support routines:

CGI Interface

```
addCGI, cgiConnect, cgiExecute, cgiGetFile, cgiPutFile
```

These routines are further described in the Support Routines section beginning on page 16.

Database Support

To include support for database requests to your command shell, add this line to xx_impl.pl:

```
require "cs_db.pl";
```

You must call *dbInit* from your *localInit* routine and call *dbExit* from your *localExit* routine.

> NOTE: The current implementation only supports access to a MySQL database via MAMP on Macintosh OS X.

User Commands

This file will add *db* mode with the following built-in commands available to users of your command shell.

Command	Description
info	Prints general information about the currently open database
open	open database [user] [password] Opens the <database> with permissions assigned to <user>
query	query sql Executes SQL query and shows results
table	table name [--dump] Lists fields or dumps contents of the table <name>
tables	Lists tables in the database

Support Routines

This file defines the following support routines:

Database

```
dbDelete, dbDo, dbFetch1Hash, dbFetchArray, dbFetchHash, dbInsertRow,
dbIsOpen, dbIsRunning, dbOpen, dbPseudoField, dbQuote, dbSelect,
dbSelectHash, dbShowQueryResults, dbTableHash, dbUpdateHashes
```

These routines are further described in the Support Routines section beginning on page 16.

Examples

The examples that follow query the *employees* sample database, which can be downloaded from https://dev.mysql.com/doc/employee/en/

The following code will open the database *employees*, search the table *employees* for all rows where the field *last_name* begins with "*Zh*", and the extract a hash with all fields for the first row in that result (retrieved using the key *emp_no*).

```
dbOpen("employees","user","pw")
@res = dbFetchHash(dbSelect(["emp_no"],"employees",
  ["last_name","REGEXP",qr/^Zh/],"emp_no"));
%employee = dbFetch1Hash(dbSelect("*","employees",
  [["emp_no","=",$res[0]->{emp_no}]]));
```

The following code will search the table *dept_emp* for all rows where the field *dep_no* matches is "d005".

```
@res = dbFetchHash(dbSelect("emp_no","dept_emp",
  ["dept_no","=","d005"],"emp_no"));
```

The following code constructs an empty hash for an entry in the table *employees*, sets the field *last_name* to "Zhiwei", and then retrieves all rows where the last name matches that value.

```
%employee = dbTableHash("employees");
$employee{last_name} = "Zhiwei";
@res = dbFetchHash(dbSelectHash("employees",\%employee));
```

The following code changes the gender of *emp_no* 99999 to "F".

```
%employee = dbFetch1Hash(dbSelect("*","employees",["emp_no","=","99999"]));
%setEmployee = %employee;
$setEmployee{gender} = "F";
dbUpdateHashes("employees","emp_no",\%employee,\%setEmployee);
```

The following code adds a row to the table *employees*.

```
%employee = dbTableHash("employees");
$employee{emp_no} = "999999";
$employee{birth_date} = "1940-04-04";
$employee{first_name} = "Tony";
$employee{last_name} = "Lewis";
$employee{gender} = "M";
$employee{hire_date} = "2017-08-16";
dbInsertRow("employees",\%employee);
```

The following code deletes that just inserted row from the table *employees*.

```
dbDo(dbDelete("employees",["emp_no","=","999999"]));
```

Network Support

To include support for HTTP requests to your command shell, add this line to xx_impl.pl:

```
require "cs_network.pl";
```

Note that cs_network.pl uses my NetRequests.pm module. You will find a copy of the latest version of this module in the lib subdirectory of the command shell directory.

You must call *networkInit* from your *localInit* routine and call *networkExit* from your *localExit* routine. Note that the *networkInit* call can be used to initialize a number of networking parameters such as the user agent.

User Commands

This file will add *network* mode with the following built-in commands available to users of your command shell.

Command	Description
get	get url [file] Retrieve a *url* and optionally store the results in *file*.
head	head url Retrieve headers for a *url*.
lookdown	lookdown tag [attributes] Look through the last retrieved web page for nodes with matching content.
select	select # Select a node for further processing.
ping	ping site Check connectivity for *site*.
top	Reset the search to the topmost node of the web page.
view	view # View the indicated node.

The following built-in debugging commands are also available.

Command	Description
cookie	Displays the cookie jar.
net	net save path [--headers] [--full] Displays information about the last response or saves the contents to a file.

Support Routines

This file defines the following support routines:

HTML Content

`html2text, htmlGetSelected, htmlParse, htmlParseFile, htmlTag, htmlTagClass, htmlTagId`

HTTP Requests

`httpAddHeader, httpGet, httpHead, httpPost, httpPostContent, httpSetCookie, relativeURL, resp`

These routines are further described in the Support Routines section beginning on page 16.

Examples

The following code will retrieve a web page, parse the content, and extract all the anchors.

```
my $RESP = htmlGet("http://www.mysite.com/index.html")
my $tree = htmlParse($RESP)
my @a = htmlTag($tree, "a");
```

The following code will retrieve a web page, parse the content, find a with a class of "mainBody" and then within that find a <table> with an id of "results".

```
my $RESP = htmlGet("http://www.mysite.com/index.html")
my $div = htmlTagClass($RESP, "div", "mainBody")
my $table = htmlTagId($div, "table", "results")
```

XML Support

To include support for XML support to your command shell, add this line to xx_impl.pl:

```
require "cs_xml.pl";
```

You must call *xmlInit* from your *localInit* routine and call *xmlExit* from your *localExit* routine.

User Commands

This file will add *xml* mode with the following built-in commands available to users of your command shell to explore XML content.

Command	Description
open	open path Reads the file *path* for processing.
child	child tag [n] Descends to the *n*th instance of *tag*.
doc	Displays the document type
dump	Dumps the XML structure
info	Displays information about an open XML file
node	Displays the type, attributes and children of the current node
parent	Ascends to the parent node
pop	Pops the top of the stack as the current node
push	Pushes the current node onto the top of the stack
shift	Shifts the current node onto the bottom of the stack
text	Displays the text of the current node
top	Ascends to the top of the XML document
unshift	Unshifts the bottom of the stack as the current node

Example

```
mode xml
open test.xml
child document
child body
child p 3
dump
push
top
node
pop
child r
dump
```

Support Routines and Variables

The command shell provides a number of variables and routines that are intended to make life easier for a shell developer.

Support Routines by Category

CGI Interface (requires "cs_cgi.pl")

`addCGI, cgiConnect, cgiExecute, cgiGetFile, cgiPutFile`

Data Management

`loadData, loadNamedData, storeData, storeNamedData`

Data Processing

`compare, copyArray, copyHash, csv, csvInit, exportData, exportStorage, exportToFile, importData, importFromFile, makeCopy`

Database (requires "cs_db.pl")

`dbDelete, dbDo, dbFetch1Hash, dbFetchArray, dbFetchHash, dbInsertRow, dbIsOpen, dbIsRunning, dbOpen, dbPseudoField, dbQuote, dbSelect, dbSelectHash, dbShowQueryResults, dbTableHash, dbUpdateHashes`

Date/Time Processing

`fmtDate, timeDiff`

Debugging

`addDebugFlag, array, debug, debugColor, debugger, dumpVar, dumpVarValue, first, getDebugFlag, getMainMode, resp, setDebugFlag`

Error Messages

`abort, complain, complainTrace, unknownCommand`

File / Directory Processing

`doDir, getFileList, getHome, getPath, getWD, pathFull, pathToSystem`

Formatted Output

`chopTable, comma, pagedFile, pagedOutput, pagedTable, showTable`

HTML Content (requires "cs_network.pl")

`html2text, htmlGetSelected, htmlParse, htmlParseFile, htmlTag, htmlTagClass, htmlTagId`

HTTP Requests (requires "cs_network.pl")

`httpAddHeader, httpGet, httpHead, httpPost, httpPostContent, httpSetCookie, relativeURL`

Information

`codeToFile, codeToLocation, codeToName, collectCode, getOS, getTermHeight, getTermWidth, toc`

Library Management

`findLibrary, loadLibrary`

Numeric Processing

`max, min`

Shell Command

`addAbout, addAlias, addCommand, addHelp, addHelpGroup, addMode, initHelp, setHistory, setOptionalUndef`

String Processing

`displayLength, safeString, safeSubstr, singPlural, sortAlpha, sortAlphaIC, testString, trueLength, wrapLine, wrapText`

Support Routines

`cgiExit, cgiInit, dbExit, dbInit, networkExit, networkInit, xmlExit, xmlInit`

User Input

`getNumber, getNumberList, getReply, getReplyInt, yesNo`

Support Routines in Alphabetical Order

abort

`abort` *`text, …`*

Prints *text* in red on STDERR and exits the program.

addAbout

`addAbout` *`text`*

Adds a description of the command shell.

Note: This call is automatically created by the customization script.

addAlias

`addAlias` *`command, aliases, mode`*

Adds *aliases* for a *command*. The comma separated list of aliases can apply to a single *mode* or every mode by specifying "*". Example:

```
addAlias("quit","q,bye,exit","*");
```

addCGI

`addCGI` *`name, url, user, password, wd, [alias]`*

Creates a CGI connection for server *name*, where *url* implements cgi-shell on the server, *user* and *password* are the authentication values to connect to the server, *wd* is the initial working directory on the server, and *alias* is a file that contains aliases for the CGI environment.

If *user* is undefined authentication will be disabled. Otherwise, if *password* is the null string the user will be prompted to enter the password when the connection is activated.

The current implementation of alias processing does not include argument substitution.

addCommand

`addCommand` *`name, syntax, function, [helpGroup, helpSyntax, helpText, mode, [helpExtra]]`*

Adds a command with *name* and *syntax* to invoke *function* from *mode* (or the most recently added mode); a mode of "*" will make the command available in every mode. If specified, help for the command will be added to *helpGroup* with the specified *helpSyntax* and *helpText*. Additional lines of help text can be specified with *helpExtra*. Examples:

```
addCommand("spaz","Ss",\&cmdSpaz);
```

adds the spaz command, which takes one or two strings as input and calls the *cmdSpaz* function with two arguments. This command does not appear in the help text.

```
addCommand("spaz","Ss",\&cmdSpaz,"main",
  "spaz foo [bar]",
  "Displays <foo> information for <bar> (or all)");
```

adds the same command with help text added to the “main” help group.

Syntax

The syntax argument consists of the concatenation of zero or more short strings (either a single letter or "-" followed by a single letter); in these strings, the letters indicate the data type of the argument and the case indicates whether the argument is required or optional. Possible values are:

Value	Description
p	Command is a prefix and everything following the prefix is a single argument (must appear alone in the syntax)
Positional arguments	
F	Required file path (may not appear after any optional arguments)
I	Required integer (may not appear after any optional arguments)
S	Required string (may not appear after any optional arguments)
f	Optional file path
i	Optional integer
s	Optional string
*	Unclaimed text (must appear after all positional arguments)
Switches	
-b, -B	optional boolean switch defaulting to 0 (for -b) or 1 (for -B). User negates -B by using --no-SWITCH (see Example syntax #2)
-X	where X is one of [iIsS]; optional switch followed by a string or integer

If any switches appear, they are separated from the syntax string and each other by colons. Arguments are passed to the implementing function in the order they are defined in the syntax.

If the command accepts optional integers or strings, the variables will be set to 0 or the null string if the user does not specify a value. To change this behavior globally and have the variables set to undef instead, call setOptionalUndef(1) or to change the behavior for just this command, prepend "undef," to the syntax argument . For example: "undef,Ss".

If the command accepts unclaimed text, anything that does not match the syntax will be accumulated into an array of strings that is passed into this argument.

Example syntax # 1

```
Sss-b-I:--silent:--wait
```

indicates that the command has three positional arguments and two switches where --silent is a boolean (the argument changes from 0 to 1 when specified) and --wait must be followed by an integer. The routine would be called with five arguments and might be implemented as:

```
sub cmdFoo
{
  my ($path, $to, $type, $silent, $wait) = @_;
  # code to implement the command
}
```

If the user input was "foo /path/to/file", the function would be called as:

```
cmdFoo("/path/to/file", "", "", 0, 0)
```

If the user input was "foo /path/to/file --silent newFile --wait 60", the function would be called as:

```
cmdFoo("/path/to/file", "newFile", "", 1, 60)
```

Example syntax # 2

```
Is-b-B:--silent:--wrap
```

indicates that the command has two positional arguments (a required integer and an optional string) plus a switch named –silent. If this were for your *test* command, then any of the following would be considered valid inputs:

```
test 1
test 1 two
test --silent 1 --no-wrap
test --silent 1 two
test 1 --silent two
test 1 two –silent --no-wrap
```

but the following would be considered invalid inputs:

```
test two
test two 1
test --quiet 1
```

---

addHelp — `addHelp` *`group, command, text`*

Adds additional help text to a help group. Note that help lines are displayed in the order they are added. If you want the help text to begin with an explanation followed by the commands in the group, add the explanation before calling *addCommand*.

If *command* and *text* are both blank, an empty line is added; otherwise, if *command* is empty, the *text* will span the entire line.

---

addHelpGroup — `addHelpGroup` *`mode`*

Adds a help group for *mode*.

Note: The help group is automatically created by *addMode*.

---

addMode — `addMode` *`name, prefix, errorRoutine, help, prefixCommand, prefixHelp`*

Adds the mode *name*, which can be invoked from another mode using the *prefix* followed by "!". Undefined commands will invoke the *errorRoutine* (for

example, *&unknownCommand*). This automatically adds a help group and a command to enter the mode. The help text will be display for the *name* command. The *prefixCommand* and *prefixHelp* will be display for the command prefix.

Example:

```
addMode("main", "m!", \&unknownCommand,
  "Process in main mode","m!command",
  "Execute a main <command>");
```

produces the following help output:

```
m!command  Execute a main <command>
main       Process in main mode
```

Note: This call is automatically created by the customization script for the main mode of your command shell.

---

addDebugFlag

`addDebugFlag` *`flag, value`*

Adds a debug flag and sets its default value.

---

array

`array \@array`
`array` *`@array`*

Prints contents of *array* truncated to the width of the terminal.

---

cgiConnect

`cgiConnect` *`name, password`*

Connects to *name* with *password*. If the password is not supplied, the user will be prompted to enter it when the site is accessed.

---

cgiExecute

`cgiExecute` *`command, silent`*

Executes *command* on the remote server and return the results. The results will also be displayed on the terminal unless *silent* is specified.

---

cgiExit

`cgiExit`

Prepares the CGI support routines for exiting the command shell.

---

cgiGetFile

`cgiGetFile` *`source, [destination], [chmod], [replace], [gzip], [silent]`*

Copies the file *source* on the server to *destination*. The file permissions will be set to *chmod* if specified. The destination file will be replaced if *replace* is specified. The file will be compressed on the server and decompressed locally if *gzip* is specified. Output will be suppressed if *silent* is specified.

cgiInit

`cgiInit`

Initializes the CGI support routines.

cgiPutFile

`cgiPutFile` *`source, [destination], [chmod], [replace], [gzip], [silent]`*

Copies the local file *source* to the server as *destination*. The file permissions will be set to *chmod* if specified. The destination file will be replaced if *replace* is specified. The file will be compressed locally and decompressed on the server if *gzip* is specified. Output will be suppressed if *silent* is specified.

chopTable

`chopTable` *`rows`*

Formats and displays a table (see *showTable*) with output lines chopped to the terminal width.

codeToFile

`codeToFile` *`code`*

Returns defining file for the referenced code.

codeToLocation

`codeToLocation` *`code`*

Returns "line # of <file>" for the first executable line of the referenced code.

codeToName

`codeToName` *`code`*

Returns the subroutine name of the referenced code.

collectCode

`collectCode` *`\%HASH, [ref, [prefix]]`*

Collects information about code from the symbol table into the hash. The key to the hash is the code name and the value is a hash reference that contains file and line. Examples:

```
collectCode(\%CODE);
collectCode(\%CODE,"Term::");
collectCode(\%CODE,\%Term::,"Term::");
map { push @out, "$_\t$CODE{$_}->{line}\t$CODE{$_}->{file}" }
  sortAlpha(keys %CODE);
```

comma

`comma` *`value`*

Returns comma notated value (e.g., comma(1234) returns “1,234”).

compare

`compare` *`\@A, \@B`*
`compare` *`\%A, \%B`*

Compares two objects and returns 1 if they have identical contents.

complain

`complain` *`text, …`*

Prints *text* in red on STDERR.

complainTrace

`complainTrace` *`text, …`*

Prints *text* in red followed by a stack trace.

copyArray

`copyArray \@array`

Returns a reference to a safe copy of the array. Any references to data in the copy will point to an object of the same type and value, but changes in the copy will not affect the original.

copyHash

`copyHash \%HASH`

Returns a reference to a safe copy of the hash. Any references to data in the copy will point to an object of the same type and value, but changes in the copy will not affect the original.

csv

`csv $CSV`

Returns a HASH reference for the next line of data (or undef if complete). Example:

```
my $CSV = csvInit($path);
my @out;
push @out, join("\t",@{$CSV->{fields}});
while (my $DATA = csv($CSV))
{
  push @out,
    join("\t", map { $DATA->{$_} }@{$CSV->{fields}});
}
showTable(@out);
```

csvInit

`csv path`
`csv \@lines`

Initializes CSV processing for either a file or an array of data returning a HASH reference to be passed to csv. The key/value pairs in the HASH are:

fields An array of the fields in the CSV data

done A boolean value that is true once all data has been processed.

dbDelete

`dbDelete table, [where]`

Constructs a DELETE statement to delete rows from *table* according to *where* conditions. WARNING: If *where* is not included, the resulting DELETE statement will delete all rows from *table*.

Pass the resulting statement to dbDo to execute it.

dbDo

`dbDo sql`

Processes *sql* statement and returns true if successful.

dbExit

`dbExit`

Prepares the database support routines for exiting the command shell.

dbFetch1Hash

`dbFetch1Hash sql`

Processes *sql* statement and returns a hash corresponding to the first result.

dbFetchArray `dbFetchArray sql`

Processes *sql* statement and returns an array of the results.

dbFetchHash `dbFetchHash sql`

Processes *sql* statement and returns an array of hashes of the results.

dbInit `dbInit`

Initializes the database support routines.

dbInsertRow `dbInsertRow table, \%HASH`

Inserts a row into *table* using the values in the hash. Note that dbTableHash can be used to create an empty hash corresponding to the fields of *table*.

dbIsOpen `dbIsOpen name`

Returns true if the database *name* is open.

dbIsRunning `dbIsRunning`

Returns true if the MySQL server is running.

dbOpen `dbOpen database, [user], [password]`

Opens the database with permissions assigned to *user*.

dbPseudoField `dbPseudoField table, field, key, type, default, null`

Defines a pseudo field that is added to hashes for the *table* that has the name *field*, and the specified *type*, and *default* value. The field may be null if *null* is "YES". Note that the field is not set automatically when values are fetched from the database. Example:

```
dbPseudoField("library","dvd_title","","varchar(255)",
  undef,"YES");
map {
  my $title = $_->{title};
  my %library = dbTableHash("library");
  $library{dvd_id} = $_->{id};
  my @in_library = dbFetchHash(dbSelectHash("library",
    \%library));
  map { $_->{dvd_title} = $title } @in_library;
} @dvd;
```

dbQuote `dbQuote value`

Quotes *value* so that it can be used in a SQL statement.

dbSelect `dbSelect [fields], table, [where], [order]`

Constructs a SELECT statement to retrieve *fields* from *table* according to *where* conditions sorted by *order*.

If *where* is specified, a WHERE clause will be included; *where* must either be an array of three elements or an array of arrays of three elements each. The three elements are *field* name, comparison operator, and *value*. Examples of the comparison operator are "<", "=", and "REGEXP". The *value* element may be a regular expression.

Pass the resulting statement to dbFetch1Hash, dbFetchArray, or dbFetchHash.

---

dbSelectHash `dbSelectHash table, \%HASH`

Constructs a SELECT statement to retrieve rows of *table* that match the values in the hash. Note that dbTableHash can be used to create an empty hash corresponding to the fields of *table*.

Pass the resulting statement to dbFetch1Hash, dbFetchArray, or dbFetchHash.

---

dbShowQueryResults `dbShowQueryResults [table], result`

Prints the values of all rows that appear in the result.

---

dbTableHash `dbTableHash table`

Returns a hash containing all the fields of the *table* set to default values.

---

dbUpdateHashes `dbUpdateHashes table, key, \%OLD, \%NEW`

Updates a row in *table* where the primary *key* matches the value in the OLD hash. Any fields in the NEW hash that do not match the OLD hash are updated.

---

debug `debug text, …`

Prints debugging output to the terminal. If the *text* is not already colorized, color will be added based on the source file of the calling routine. (See *debugColor*.)

---

debugColor `debugColor color, [source]`

Sets the debugging color for the specified *source* file (or the caller's source file if none.) Example:

```
debugColor("blue",codeToFile(\&NetRequests::new));
```

---

debugger `debugger evaluator`

Invokes the debugger (see page 3) with an evaluator to provide access to locally-scoped variables.

---

displayLength `displayLength text`

Returns the number of characters that will be displayed on the terminal if the string is printed (excluding any colorizing characters). Example:

```
displayLength("$COLOR{blue}abcd$COLOR{reset}")
```

returns 4.

doDir

```
doDir dir, walk, routine
doDir dir, walk, [rules]
```

Traverses a directory and takes designated actions on appropriate objects. Returns the number of items on which action was taken. Emits an error message and returns undef if the *rules* cannot be successfully parsed.

Rules select the items to be processed and specify the actions to be performed on those items:

dir	alternative to positional *path* to the directory
walk	alternative to the positional *walk* flag
accept	if any accept conditions are found, this rule is applied
reject	… unless any reject condition is found
run	routine to be invoked to process the item
descend	routine to be invoked when descending into a subdirectory
ascend	routine to be invoked when ascending from a subdirectory
data	additional data to be passed to the *run*, *descend*, and *ascend* routines

Accept and reject conditions are either regular expressions or strings that are some combination of the characters "f", "l", and "d" for files, links, and directories respectively.

The ascend, descend, and run routines will be invoked as follows:

```
routineToCall($root,$relPathToItem,@data)
```

where *$root* is the path passed to doDir and *$relPathToItem* is the relative path to the individual item. The full path for each item is "*$root/$relPathToItem*".

Examples

Print the name of each file, link, and directory in the current directory:

```
doDir(".",0,sub { print join("/",@_),"\n" })
```

Print the name of each file in the current directory and all subdirectories:

```
doDir(".",1,
  [accept => "f",
    run => sub { print "file: ",join("/",@_),"\n" }])
```

Print the name of each file in the current directory and all subdirectories:

```
doDir(".",1,
  [reject => "dl",
    run => sub { print "file: ",join("/",@_),"\n" }])
```

Run &processText on all *.txt files that do not begin with "foo":

```
doDir(dir => ".",
  walk => 1,
  accept => qr/\.txt$/,
  reject => qr/^foo/,
  run => \&processText,
  data => \%TEXT)
```

Print the names of *.txt and *.pl files:

```
doDir(dir => ".",
  walk => 1,
  [ accept => qr/\.txt$/,
    run => sub { print "Text: ",join("/",@_),"\n" }],
  [ accept => qr/\.pl$/,
    run => sub { print "Perl: ",join("/",@_),"\n" }])
```

Collect all sub-directories into @dirs array:

```
doDir(dir => $wd,
  walk => 1,
  [descend => sub { push @dirs, join("/",@_) }]);
```

---

dumpVar

`dumpVar` *`var, [descend]`*

Prints contents of *var*. If *descend* is true then the nested contents will also be printed.

---

dumpVarValue

`dumpVarValue` *`var`*

Prints value of *var*.

---

exportData

`exportData` *`filehandle, label, data`*

Exports *data* to an open file. Example:

```
exportData($fh,"\%HASH",\%HASH)
```

The export routines provide an operating system neutral mechanism to move data between machines.

---

exportStorage

`exportStorage` *`storage, filename`*

Exports data from *storage* to *filename*. Example:

```
exportStorage("/path/to/storage","/path/to/file")
```

---

exportToFile

`exportToFile` *`filename, label, data`*

Exports *data* to *filename*. Example:

```
exportToFile("/path/to/file","\@array",\@array)
```

If *filename* has a .tgz extension, the data will be written to a temporary file with an extension of .exp and then that file will be compressed to the requested name. For example, if the path is "foo.tgz", the data will first be written to "foo.exp" and that file will be compressed to "foo.tgz".

findLibrary

```
findLibrary name
```

Returns an array of paths to a system library.

first

```
first %HASH
first \%HASH
```

Prints contents of the first key/value pair of *HASH*.

fmtDate

```
fmtDate format, tm, [gmt]
```

Formats the number of non-leap seconds since January 1, 1970 into a string (using local time unless gmt is non-zero). Example:

```
fmtDate("yyyy-dd-mm",time());
```

Format strings:

am	either "am" or "pm"	m	single digit month: 1
AM	either "AM" or "PM"	mm	two digit month: 01
d	single digit day: 1	mmm	short month name: Jan
dd	two digit day: 01	mmmm	long month name: January
ddd	short weekday: Mon	n	single digit minute: 1
dddd	long weekday: Monday	nn	two digit minute: 01
h	single digit 24-hour clock hour: 1	s	single digit second: 1
hh	two digit 24-hour clock hour: 01	ss	two digit second: 01
hhh	single digit 12-hour clock hour: 1	yy	two digit year: 01
		yyyy	four digit year: 2001

getDebugFlag

```
getDebugFlag flag
```

Returns the current setting for the named debug flag.

getFileList

```
getFileList [directory,] match
```

Returns an array of files that *match* specified criteria from a directory. Examples:

```
getFileList("*.pl")
getFileList("images/*.jpg")
getFileList("images","*.jpg")
getFileList("images",qr/\.(?:bmp|gif|jpeg|jpg|png)$/i)
```

getHome

```
getHome
```

Returns the user's home directory.

getMainMode

```
getMainMode
```

Return the name of the main mode.

getNumber

```
getNumber prompt, low, high
```

Prompts the user for input and returns the result, which will be a number between low and high inclusive, or undef if the user did not supply a response.

getNumberList `getNumberList` *`prompt, low, high`*

Prompts the user for input and returns the result, which will be a list of numbers between low and high inclusive, or undef if the user did not supply a response.

getOS `getOS`

Returns a string indicating the operating system (one of cygwin, macosx, unix, or windows).

getPath `getPath` *`path`*

Returns the full path for a relative path; recognizes "~" as the user's home directory. Example:

```
getPath("~/.bash_profile")
```

getReply `getReply` *`prompt, [allowed, [password]]`*

Prompts the user for input, compares that input against what is allowed, and returns a valid result or undef if the user did not supply a response. If *password* is specified, the user's input is not echoed to the terminal.

The *allowed* string consists of a series of valid responses separated by semicolons. A response can map multiple user inputs onto a single return value by including the return value, an equal sign, and comma-separated alternatives. A default response is indicated by a leading "!". The following are three alternatives for a yes/no question:

```
"yes;no"
"yes=yes,y;no=no,n"
"yes=yes,y;!no=no,n"
```

The first alternative accepts only "yes" or "no" and returns undef if the user does not enter anything. The second alternative accepts either "yes" or "y" for "yes" and "no" or "n" for "no" and returns undef if the user does not enter anything. The third alternative is like the second except that it will return "no" if the user does not enter anything.

The debugger can be invoked during any call to getReply (including calls to getNumber, getNumberList, and yesNo) by entering ":debugger" as the response.

getReplyInt `getReplyInt`

Returns a boolean value indicating if the user interrupted the most recent call to getReply (or one of its variations) with ^C.

getTermHeight `getTermHeight`

Returns the height of the user's terminal.

getTermWidth `getTermWidth`

Returns the width of the user's terminal.

getWD `getWD`

Returns the current working directory.

html2text `html2text` *html*

Returns *html* with character codes converted to human-readable text.

htmlGetSelected `htmlGetSelected tree, name`

Returns the text for a selected option. Example:

```
my $form = htmlTagId($tree,"form","request");
my $selected = htmlGetSelected($form,"rate");
```

htmlParse `htmlParse` *content*

Returns a parsed tree for the content. Example:

```
my $RESP = httpGet("http://www.example.com/index.html")
$RESP->{tree} = htmlParse($RESP);
```

htmlParseFile `htmlParseFile` *filename*

Returns a parsed tree for the contents of a file.

htmlTag `htmlTag` *content, tag*

Searches through *content* for a matching *tag*. The *content* can be the response to a request (such as httpGet), a parsed Tree (such as returned by htmlParse), the result of a previous search request (such as htmlTag) or raw HTML content. This call is functionally equivalent to:

```
$tree->look_down('_tag',$tag);
```

htmlTagClass `htmlTagClass` *content, tag, id*

Searches through *content* for a matching *tag* with a class of *id*. This call is functionally equivalent to:

```
$tree->look_down('_tag',$tag,'class',$id);
```

htmlTagId `htmlTagId` *content, tag, id*

Searches through *content* for a matching *tag* with an ID of *id*. This call is functionally equivalent to:

```
$tree->look_down('_tag',$tag,'id',$id);
```

httpAddHeader `httpAddHeader` *name, value*

Adds a header for the next request.

httpGet — `httpGet url`

Sends GET request for *url* and returns reference to httpResponse. Example:

```
my $RESP = httpGet("http://www.example.com/index.html");
```

httpHead — **`httpHead url`**

Sends HEAD request for *url* and returns reference to httpResponse. Example:

```
my $RESP = httpHead("http://www.example.com/index.html");
```

httpPost — `httpPost url, content`

Posts *content* to *url* and returns reference to httpResponse. If *content* is an array, it will be processed by httpPostContent. Example:

```
my $RESP = httpPost("http://www.example.com/index.php",
  $content);
```

httpPostContent — `httpPostContent name => value, …`

Formats name/value pairs to pass as content to a POST request. The values are sanitized replacing the characters space, '%', '&', and '=' with %20, %25, %26, and %3D respectively. Example:

```
httpPostContent([
  name    => "Tony",
  company => "Lewis & Sons",
  motto   => "Unity=Power"])
```

returns "name=Tony&company=Lewis%20%26%20Sons&motto=Unity%3DPower".

httpSetCookie — `httpSetCookie key, value, path, domain, port, secure, maxage`

Adds a cookie to the cookie jar for future requests.

initHelp — `initHelp name, short, command, help`

Initializes help processing of the command shell with a descriptive *name*, a *short* form of the name, a phrase to describe a single *command*, and the name of the *help* routine (typically "help").

Note: This call is automatically created by the customization script.

importData — `importData filehandle`

Imports data from an open file.

importFromFile — `importFromFile filename`

Imports data previously exported by exportData, exportToFile, or exportStorage from *filename*. Example:

```
importFromFile("/path/to/exportFile");
```

If *filename* has a .tgz extension, the file will be decompressed to a temporary file and the data will be imported from the temporary file.

loadData

```
loadData path container …
```

Load data previously stored in *path* by storeData into one or more containers. A container will be an an array, a hash, or a scalar. The quantity and data types of the containers must match the call to storeData.

loadLibrary

```
loadLibrary name, reason
```

Loads a Perl library and records the reason it was loaded.

loadNamedData

```
loadNamedData path [container …]
```

Load named data previously stored in *path* by storeNamedData into one or more containers.

makeCopy

```
makeCopy \@A, \@B
makeCopy \%A, \%B
```

Makes a safe copy of *B* in *A*. Any references to data in the copy will point to an object of the same type and value, but changes in the copy will not affect the original.

max

```
max number, …
```

Returns the largest of a group of numbers.

min

```
min number, …
```

Returns the smallest of a group of numbers.

networkExit

```
networkExit
```

Prepares the network support routines for exiting the command shell.

networkInit

```
networkInit name, value, …
```

Initializes the network support routines. Accepts the following name/value pairs to control network processing and to optionally set information passed to a remote site in a request:

accept	Sets value for the HTTP Accept header (default: */*)
accept-lang	Sets the value for the HTTP Accept-Language header (default: en-us)
block	Array of domain names that are blocked; requests for those domains will automatically fail with a status of 500
base	Default base
redirect	Set to 0 to suppress automatic processing of redirect requests
referer	Sets value for the HTTP Referer header for the first request
user-agent	Sets value for the HTTP User-Agent header (default: Mozilla)

	Any other name/value pairs will be passed to LWP::UserAgent. Note that setting the referer header will automatically change to the last URL processed for subsequent requests.
pagedFile	`pagedFile` *`filename`* Prints contents of a file one page at a time.
pagedOutput	`pagedOutput` *`\@text`* `pagedOutput` *`@text`* `pagedOutput` *`filehandle`* Prints output to screen one page at a time.
pagedTable	`pagedTable rows` Formats and displays a table (see *showTable*) one page at a time.
pathFull	`pathFull` *`filename`* Returns the fully-qualified path to *filename*. If the *filename* is a link, the result will be the fully-qualified path to the target of the link.
pathToSystem	`pathToSystem` *`filename`* Returns *filename* with escaped character sequences safe to pass to operating system commands.
relativeURL	`relativeURL base, [parent], url` Returns *url* relative to the *base* and the *parent* page.
resp	`resp` *`[\%RESP]`* Prints contents of the HTTP response.
safeString	`safeString` *`text`* Returns copy of *text* safe to pass to print. Result is either an ASCII or UTF-8 string with no wide characters.
safeSubstr	`safeSubstr` *`text,pos[,n]`* Returns a substring *n* characters long starting at *pos* treating UTF-8 character sequences as a single character.
setDebugFlag	`setDebugFlag` *`flag, value`* Sets the named debug flag to *value*.
setHistory	`setHistory` *`filename`* Sets the file for saving command history between invocations of the command shell. Note: This call is automatically created by the customization script.

setOptionalUndef `setOptionalUndef` *`setting`*

Globally sets the behavior for the value of unspecified optional command arguments. If *setting* is non-zero, undefined command arguments will return *undef*, otherwise they return a null string or zero (depending on the type of the argument).

showTable `showTable` *`rows`*

Formats and displays a table. Each row of output will take one of two forms:

```
"<field>\t<field>\t…"
"\f<text>"
```

A row that begins with "\f" will span the entire row; all other rows will contain the fields separated by tabs aligned into columns. By default, each field will be left aligned and as wide as necessary to fit the field.

Formatting can be fine-tuned by passing instructions as the first line. The formatting line must start with "\a" and contain information for each column formatted roughly as %[[n.m][,inst...]] where *n* is the minimum width, *m* is the precision (affects only floating-point numbers), and *inst* may be one or more of the following:

center	Center the text
chop:w	Chop the field at <w> characters
indent	If text exceeds the terminal width, indent the wrapped text to this column
comma	Add comma separator to numeric values
left	Left justify the text (default unless <m> is specified)
right	Right justify the text (default if <m> is specified)
tab:n	Number of spaces following this column (default: 1)
wrap:w	If this column exceeds <w> characters, display the text as a full line and wrap the remaining columns to the next line

A field with no special instructions is indicated by '%'

Example Fields

%5	Formats 'abc' as 'abc '
%5.2	Formats pi as ' 3.14'
%8,center	Formats 'abcd' as ' abcd '
%8,right	Formats 'abcd' as ' abcd'
%chop:7	Formats 'abcdefghijk' as 'abcdefg'
%comma	Formats 12345 as '12,345'

Example

```
showTable("\a%%5,right%chop:7","1.\tabc\texplanation",
          "2.\twxyz\tshort")
```

produces:

```
1.   abc explana
2.  wxyz short
```

singPlural

`singPlural` *`n, singular, plural`*

Returns the singular form if n is 1; otherwise returns the plural form. The form may include "%d", which will be replaced by *n*. Examples:

```
singPlural($n,"just 1 file","many files")
singPlural($n,"%d file","%d files")
```

sortAlpha

`sortAlpha` *`@array`*

Returns *array* sorted alphabetically.

sortAlphaIC

`sortAlphaIC` *`@array`*

Returns *array* sorted alphabetically without regard to case.

storeData

`storeData` *`path, container, …`*

Store one or more containers in *path*. A container will be an array, a hash, or a scalar. For example:

```
storeData(".data", \@users, \%INFO, \$date);
```

storeNamedData

`storeNamedData` *`path, name, container, …`*

Store one or more named containers in *path*. A container will be an array, a hash, or a scalar. If the names are valid Perl variable names (as in the second example below), the data can be reloaded into those variables by loadNamedData without specifying the containers. Examples:

```
storeNamedData(".cs",
  config => \%CONFIG,
  list =>   \@list,
  info =>   \$info);
storeNamedData(".cs",
  "\%CONFIG", \%CONFIG,
  "\@list",   \@list,
  "\$info",   \$info);
```

testString

`testString` *`text`*

Tests contents of string. Returns value from 0 to 7 where 1 bit is on if *text* contains ASCII characters, 2 bit is on if *text* contains UTF-8 characters, and 4 bit is on if *text contains* wide characters.

timeDiff	`timeDiff a, b` Returns a string describing the time that has elapsed (or will elapse) between *a* and *b*.

toc	`toc path, [pageLength, pageWidth]`
	`toc \@path, [pageLength, pageWidth]`
	
	Generates a Table of Contents for a source file (or a collection of source files).

trueLength	`trueLength text` Returns the number of UTF-8 characters in *text*.
unknownCommand	`unknownCommand` Prints "What?" in red. You may use this routine as a mode error routine.
wrapLine	`wrapLine text` Word wraps a line of output so that it will fit on the terminal.
wrapText	`wrapText text, width` Word wraps a line of output so that it will fit in the specified width.
xmlExit	`xmlExit` Prepares the XML support routines for exiting the command shell.
xmlInit	`xmlInit` Initializes the XML support routines.
yesNo	`yesNo prompt` Prompts the user for a yes/no response; returns 1 for yes, 0 for no, or undef if the user did not supply a response.

Support Variables

The scalar **$debug** is available to control debugging output; it will have a value between 0 and 3 inclusive and can be changed at runtime using the *debug* command.

The hash **%COLOR** is available for use by user-developed code to colorize terminal output. For example:

```
print "$COLOR{blue}This text will appear in blue.$COLOR{reset}\n";
```

The hash **%VT100** is available for use by user-developed code to control a VT100 compatible terminal. For example:

```
print $VT100{nl};
```

Using the support routines in other scripts

The file cs_fn.pl contains many support routines and it has been written so that it can be added to scripts other than the command shell. (For example, customize.pl requires it.) To use these routines, simply add the following to your code:

```
require "cs_fn.pl";
```

Note that any routines that begin with “cs_” are not intended to be called directly by your code. If you find a useful “cs_” routine, you should create a wrapper function in cs_fn.pl to isolate your command shell from future script changes.

Implementation Details

The launch file (xx.pl) has special code to load files from the directory where it is located (even when files in that directory would not ordinarily be loaded by Perl. In addition, the command shell will look for a directory named “lib” and add it to the Perl include array. It will add the first match from the following list:

- a directory specified in the environment variable named “lib”
- a directory named lib in the current working directory (./lib)
- a directory named lib in the user’s home directory (~/lib)

User commands are read using a variation of Term::ReadLine. The command shell will use the first library found in the following list:

- Term::ReadLine::Gnu
- Term::ReadLine::Perl
- Term::ReadLine

Customization

The customization script will ask you a series of questions and will create a customized command shell based on the responses. For most questions a default response will be displayed within brackets, e.g., [Sample]. If you want to accept the default answer, simply press the [Enter] or [Return] key on your keyboard. You can jump to a previously encountered step by entering :# as your reply to any question. For example, from step 6, you can enter *:2* to jump back to step 2 and enter a new short name and then enter *:6* to return to step 6.

1. Descriptive Name

Prompt What will you call you command shell?

Give your command shell a descriptive name. Choose something that describes the function(s) that the shell implements. You might want to use initial caps in the descriptive name, e.g., “Fancy Command Shell”.

Usage
- Comment block at the start of each file.
- Copyright notice
- Output of help command

2. Short Name

Prompt What is the short form of the name?

Choose a short form of the descriptive name. This will typically be a single word and should not be capitalized.

Usage
- Output of help command

3. Command Description

Prompt What is the phrase that describes one of your commands?

Choose a phrase to describe your commands. This is typically the short name preceded by an article, e.g., “a fancy” or “an installer”.

Usage
- Output of help command

4. Help Command

Prompt What command will users enter for assistance with commands?

Enter the name of the internal help command, e.g., “help”.

Usage
- Name of the help command

5. Help Group

Prompt What help group will the **Short Name** commands be assigned to?

Commands within the shell are grouped into help groups; you may define multiple help groups in your shell, but enter the main help group now.

Usage
- Name of the help group

6. Command History

Prompt Where will the command history be stored?

The command shell remembers the commands that the user enters and stores the last 500 commands. Enter the location of the file where the command history will be stored.

Usage
- Command history file location

7. Command Mode

Prompt What will your mode be called?

Commands within the shell are grouped into modes. (Built-in modes include cgi, perl, and system.) You may define multiple modes

Usage
- Command mode for your commands

8. Command Prefix

Prompt What letter will be the prefix for commands in the "**Command Mode**" mode?

Choose a single letter of the alphabet to be used as a prefix for your commands from another mode. Note that 'c' is reserved for the cgi mode and 'p' is reserved for the Perl mode.

For example, if you choose 'm', , a user can enter `m!test` to run your 'test' command from another mode.

Usage
- Prefix for your commands

9. File Prefix

Prompt What prefix would you like to use for the files?

There are a total of three files that must share the same prefix. For example, if your prefix is 'xx', the following files are created:

xx.pl	This is the main shell that users will invoke.
xx_defs.pl	This file defines the inner workings and commands.
xx_impl.pl	This file implements your commands.

Of course, you can add other files as necessary, but that is the default set. You must use a prefix other than "cs" so that your files do not get accidentally overwritten when you upgrade to a later version of the command shell.

Usage
- Prefix of generated source file names

10. Author Name

Prompt Who is the author of this command shell?

Enter your name (or the name of your organization).

Usage
- Name of the command's author

11. Author Email

Prompt What is the author's email address?

Enter your email address.

Usage
- Email address of the command's author

12. Version Number

Prompt What is the initial version number?

Enter the initial version number; for example: 0.1 or 1.0.

Usage
- Version number of the initial version

13. Description

Prompt The **Descriptive Name** provides a set of commands to...

Enter a command description. This description will appear in a comment block of the generated source files and in the output of the 'about' command.

Usage
- A description of the command

14. Copyright Notice

Prompt What is your copyright notice?

Enter a copyright notice to appear in the generated source files. You can choose a notice of the form "Copyright 20xx **Author Name**" by entering '-'. The copyright notice will appear in a comment block of the generated source files and in the output of the 'about' command.

Usage
- Copyright notice for the command

15. Source Directory

Prompt Do you want to use the default directory?

A new directory can to be created to hold your new command files. By default this new directory will have the same name as the file prefix specified in step 9.

Usage
- Directory where generated source files will be created

Example Customization

The following is an example of running the customization script. In this example, "`[Enter]`" means to press the Enter or Return key to accept the default response.

```
This utility provides a command shell into which you can add your own commands.

To get started, answer a few questions about your the inner workings of your
command shell.

1. You need to give your command shell a name (example: "Fancy Command Shell")
What will you call you command shell? [Sample] Fancy Command Shell

2. You also need a short form of the name (example: "fancy")
What is the short form of the name? [fancy] [Enter]

3. You need a phrase that describes a single one of your commands (for example,
"a fancy")
What is the phrase that describes one of your commands? [a fancy] [Enter]

4. You may give the 'help' command a different name.
What command will users enter for assistance with commands? [help] [Enter]

5. Internally your commands will be assigned to a help group (example: "fcs").
What help group will the fancy commands be assigned to? [fcs] [Enter]
```

```
6. In order for the command history to be saved between runs of your shell, you
must specify a file name or path. If you specify a file name, the history will be
stored in the same directory as the command shell. You might also store it in the
user's home directory by preceding the file name with ~/
Where will the command history be stored? [.fcs_history] [Enter]

7. Within the shell, your commands will be collected into a separate mode.
What will your mode be called? [fcs] [Enter]

8. You need to specify a prefix to execute commands in the 'fcs' mode from
another mode. Note that 'p' is reserved and 'c', 'n', and 'x' are used by optional
command shell support routines.
What letter will be the prefix for commands in the "fcs" mode? [f] [Enter]

That takes care of the inner workings. Now let's move on to the files that will
be created. There are total of three files that must share the same prefix.
The names are of the form:
  xx.pl       This is the main shell that users will invoke.
  xx_defs.pl  This file defines the inner workings and commands.
  xx_impl.pl  This file implements your commands.

Of course, you can add other files as necessary, but that is the default set. You
must use a prefix other than "cs" so that your files do not get accidentally
overwritten when you upgrade to a later version of the command shell.

9. What prefix would you like to use for the files?
What prefix will be used for your files? [fcs] [Enter]

10. What author name do you want to appear in the source?
Who is the author of this command shell? [] Your Name

11. What is the author's email address?
What is the author's email address? [] Your Email

12. What initial version number do you want to appear in the source?
What is the initial version number? [] 1.0

13. Now describe what the command shell does. Complete the following sentence:
The Fancy Command Shell provides a set of commands to... [] do something fancy

14. Enter any copyright notice you want included (leave blank for none).
Example: Copyright 2017 Your Name <Your Email>
Enter '-' if that example is in fact how you want your copyright to read.
What is your copyright notice? [] -
```

```
15. A new directory can to be created to hold your new command files. By default
this new directory will be:
  /Users/Tony/bin/cs/fcs
If you want to create that directory, reply 'yes'. If you want to leave the files
in the current directory, reply 'no'. Otherwise, reply with the new directory.
Do you want to use the default directory? [yes] [Enter]

That is all the information needed.
Are you ready to generate your command shell? [yes] [Enter]

Your command shell has been created. Run your script using the following command:

./fcs.pl

Inside the script, enter the following commands:
about
test
help

If the output of those commands is satisfactory, start adding your own
commands. You may edit or delete the "test" command. If you are not satisfied
with the output, edit ./fcs_defs.pl or ./fcs_impl.pl.

Send feedback on the command shell and this customization script to
tlewis@exelana.com.
```

Example Test Run

The following is an example of running the command shell immediately following the customization above.

```
./fcs.pl
fcs> about
Fancy Command Shell
This command will do something fancy.

Version 1.0
Written by Tony Lewis.
Copyright 2017 Tony Lewis <tlewis@exelana.com>
Updated: August 20, 2017 at 8:17 PM

Built on command shell version 3.1 (2017-08-30 08:15:09).
For more information about the command shell, visit
http://www.exelana.com/techie/perl/cs.html
fcs> test
Hello, world!
fcs> test Tony
Hello, Tony!
```

```
fcs> help
test [name]    Initial testing command

f!command      Execute a fancy <command>
!command       Execute a system <command>
p!statement    Execute a Perl <statement>

fcs            Process in fancy mode
system         Process in system command mode
perl           Process in Perl mode

edit file      Edit <file> with _emacs
editlib name   Edit the system library <name> with _emacs
about          Display information about this script
built          Display information about the last call to showTable
cd             Change the working directory
findlib name   Display where <name> appears in the system libraries
pwd            Print the working directory
save path      Save last output in <path>
reload         Reload this program
quit           Exit this program
fcs> quit
```

Subroutine Index

Support Routines

Routine	Page	Line	File
debugger	67	3393	cs_fn.pl
displayLength	54	2798	cs_fn.pl
doDir	33	1665	cs_fn.pl
dumpVar	27	1390	cs_fn.pl
dumpVarValue	30	1517	cs_fn.pl
exportData	18	987	cs_fn.pl
exportStorage	18	951	cs_fn.pl
exportToFile	17	923	cs_fn.pl
findLibrary	44	2234	cs_fn.pl
first	28	1412	cs_fn.pl
fmtDate	24	1230	cs_fn.pl
getDebugFlag	28	1438	cs_fn.pl
getFileList	37	1910	cs_fn.pl
getHome	37	1960	cs_fn.pl
getMainMode	32	1638	cs.pl
getNumber	59	3016	cs_fn.pl
getNumberList	59	3037	cs_fn.pl
getOS	42	2135	cs_fn.pl
getPath	38	1970	cs_fn.pl
getReply	60	3098	cs_fn.pl
getReplyInt	61	3175	cs_fn.pl
getTermHeight	42	2143	cs_fn.pl
getTermWidth	42	2151	cs_fn.pl
getWD	38	1996	cs_fn.pl
html2text	3	111	cs_network.pl
htmlGetSelected	3	126	cs_network.pl
htmlParse	3	148	cs_network.pl
htmlParseFile	4	173	cs_network.pl
htmlTag	4	190	cs_network.pl
htmlTagClass	5	210	cs_network.pl
htmlTagId	5	219	cs_network.pl
httpAddHeader	5	228	cs_network.pl
httpGet	6	237	cs_network.pl
httpHead	6	249	cs_network.pl
httpPost	6	261	cs_network.pl
httpPostContent	6	274	cs_network.pl
httpSetCookie	6	300	cs_network.pl
importData	19	1040	cs_fn.pl
importFromFile	19	1008	cs_fn.pl
initHelp	33	1668	cs.pl
loadData	9	413	cs_fn.pl
loadLibrary	44	2254	cs_fn.pl
loadNamedData	10	459	cs_fn.pl
makeCopy	15	799	cs_fn.pl
max	45	2291	cs_fn.pl
min	45	2301	cs_fn.pl
networkExit	15	742	cs_network.pl
networkInit	15	724	cs_network.pl
pagedFile	46	2342	cs_fn.pl
pagedOutput	47	2356	cs_fn.pl
pagedTable	47	2416	cs_fn.pl
pathFull	38	2004	cs_fn.pl
pathNormalize	38	2020	cs_fn.pl

Routine	Page	Line	File
pathToSystem	39	2031	cs_fn.pl
relativeURL	6	310	cs_network.pl
resp	7	324	cs_network.pl
safeString	54	2808	cs_fn.pl
safeSubstr	54	2842	cs_fn.pl
setDebugFlag	28	1447	cs_fn.pl
setHistory	35	1869	cs.pl
setOptionalUndef	37	1904	cs.pl
showTable	48	2464	cs_fn.pl
singPlural	55	2875	cs_fn.pl
sortAlpha	56	2886	cs_fn.pl
sortAlphaIC	56	2894	cs_fn.pl
storeData	11	505	cs_fn.pl
storeNamedData	11	534	cs_fn.pl
testString	56	2909	cs_fn.pl
timeDiff	25	1291	cs_fn.pl
toc	43	2160	cs_fn.pl
trueLength	57	2935	cs_fn.pl
unknownCommand	31	1590	cs_fn.pl
wrapLine	57	2961	cs_fn.pl
wrapText	57	2970	cs_fn.pl
xmlExit	5	221	cs_xml.pl
xmlInit	5	213	cs_xml.pl
yesNo	61	3183	cs_fn.pl

User Commands

Command	Routine	Page	Line	File
about	cs_cmdAbout	14	704	cs.pl
built	cs_cmdBuilt	15	723	cs.pl
cd	cs_cmdCd	15	805	cs.pl
child	xmlCmdChild	2	46	cs_xml.pl
cookie	netCmdCookie	11	496	cs_network.pl
debug	cs_cmdDebug	15	832	cs.pl
doc	xmlCmdDoc	2	66	cs_xml.pl
dump	cs_cmdDump	17	910	cs.pl
dump	xmlCmdDump	2	81	cs_xml.pl
edit	cs_cmdEdit	21	1183	cs.pl
editlib	cs_cmdEditLib	23	1205	cs.pl
exit	cs_cmdQuit	26	1328	cs.pl
findlib	cs_cmdFindLib	24	1238	cs.pl
get	cgiCmdGet	6	258	cs_cgi.pl
get	netCmdGet	12	542	cs_network.pl
head	netCmdHead	12	564	cs_network.pl
help	cs_cmdHelp	24	1249	cs.pl
info	dbCmdInfo	13	649	cs_db.pl
info	xmlCmdInfo	2	87	cs_xml.pl
log	cgiCmdLog	9	416	cs_cgi.pl
lookdown	netCmdLookDown	12	582	cs_network.pl
net	netCmdNet	13	605	cs_network.pl
node	xmlCmdNode	3	108	cs_xml.pl
open	dbCmdOpen	14	672	cs_db.pl
open	xmlCmdOpen	2	96	cs_xml.pl
parent	xmlCmdParent	3	137	cs_xml.pl

Command	Routine	Page	Line	File
ping	netCmdPing	14	666	cs_network.pl
pop	xmlCmdPop	3	143	cs_xml.pl
push	xmlCmdPush	3	150	cs_xml.pl
put	cgiCmdPut	6	317	cs_cgi.pl
pwd	cs_cmdPwd	25	1313	cs.pl
query	dbCmdQuery	13	659	cs_db.pl
quit	cs_cmdQuit	26	1328	cs.pl
reload	cs_cmdReload	26	1336	cs.pl
save	cs_cmdSave	28	1440	cs.pl
select	cgiCmdSelect	8	391	cs_cgi.pl
select	netCmdSelect	14	689	cs_network.pl
shift	xmlCmdShift	3	156	cs_xml.pl
table	dbCmdTable	14	680	cs_db.pl
tables	dbCmdTables	15	719	cs_db.pl
text	xmlCmdText	3	163	cs_xml.pl
top	netCmdTop	14	702	cs_network.pl
top	xmlCmdTop	3	169	cs_xml.pl
tree	xmlCmdTree	4	175	cs_xml.pl
unshift	xmlCmdUnshift	4	184	cs_xml.pl
view	netCmdView	14	712	cs_network.pl

Internal Support Routines

Routine	Page	Line	File
_charBytes 57	2985	cs_fn.pl	
_compare 13	609	cs_fn.pl	
_csv 15	720	cs_fn.pl	
_debuggerStack	68	3469	cs_fn.pl
_debuggerWhere	68	3478	cs_fn.pl
_doDir 34	1768	cs_fn.pl	
_xmlDump 4	198	cs_xml.pl	
cs_addInternal	31	1571	cs.pl
cs_buildTableOutput	49	2480	cs_fn.pl
cs_cgiCheckCode 3	147	cs_cgi.pl	
cs_cgiConfigure 9	438	cs_cgi.pl	
cs_cgiConnect 3	132	cs_cgi.pl	
cs_cgiRequest 4	221	cs_cgi.pl	
cs_checkTree 8	346	cs_network.pl	
cs_cmdExit 24	1230	cs.pl	
cs_commandHelp	25	1290	cs.pl
cs_copyREF 16	860	cs_fn.pl	
cs_db2Date 11	491	cs_db.pl	
cs_db2Time 11	502	cs_db.pl	
cs_dbConfigure	15	754	cs_db.pl
cs_dbDebugArg	11	510	cs_db.pl
cs_dbHashFromDB 11	519	cs_db.pl	
cs_dbHashValue2db 12	538	cs_db.pl	
cs_dbQuery 12	583	cs_db.pl	
cs_dbValueFromDB 13	610	cs_db.pl	
cs_defineInternal 34	1753	cs.pl	
cs_doCGI 3	164	cs_cgi.pl	
cs_doEvalWithInterrupt 62	3193	cs_fn.pl	
cs_doLoadItem 12	561	cs_fn.pl	
cs_doLoadNamedData	10	476	cs_fn.pl

Tony Lewis' Command Shell (page 1)

User commands

about	Displays information about the script.
cd *path*	Change the working directory.
cgi	Enter "cgi" mode to send commands to a server.
exit	Return to main mode from cgi, perl or system mode
help *[topic]*	Display information about available commands
perl	Enter "perl" mode to evaluate Perl expression
pwd	Print the current working directory.
quit	Exit the command shell
system	Enter "system" mode to enter operating system commands.

Shell Developer commands

debug *[level]*	Change the debugging level (options: off on 0 1 2 3)
dump *[topic]*	Dump internal system structures full | <mode> | all | commands | help | lib
edit *file*	Edit <file>
editlib *lib*	Edit system library <lib>
findlib *lib*	Find system library <lib>
reload	Reload the script
storage *file*	Examine a file created by storeData

addCommand syntax argument

F / f	file path
I / i	integer
S / s	string
*	unclaimed
-B / -b	Boolean switch (on / off)
-I / -i	integer switch (required / optional)
-S / -s	string switch (required / optional)

Example: Fss-b-I:--silent:--wait

Directory Traversal

```
doDir   (".",0,sub { print join("/",@_),"\n" })
doDir   (".",1,
         [accept => "f",
          run => sub { print "file: ",join("/",@_),"\n" }])
doDir   (".",1,
         [reject => "dl",
          run => sub { print "file: ",join("/",@_),"\n" }])
doDir   (dir => ".",
         walk => 1,
         accept => qr/\.txt$/,
         reject => qr/^foo/,
         run => \&processText,
         data => \%TEXT)
doDir   (dir => ".",
         walk => 1,
         [ accept => qr/\.txt$/,
          run => sub { print "Text: ",join("/",@_),"\n" }],
         [ accept => qr/\.pl$/,
          run => sub { print "Perl: ",join("/",@_),"\n" }])
doDir   (dir  => $wd,
         walk  => 1,
         [descend => sub { push @dirs, join("/",@_) }]);
```

User Input (allowed)

yes;no	only accepts "yes" or "no"
yes=yes,y;no=no,n	returns "yes" for "y" and "no" for "n"
yes=yes,y;!no=no,n	null response means "no"

Initializing the Shell Command (xx_defs.pl)

addAbout(text) Add *text* to the about command

addAlias(command,alias,mode)
Adds an *alias* for *command* in *mode*

addCommand(name,syntax,routine,helpGroup,helpSyntax,helpText,mode)
Adds a command to *mode*

addHelp(group,command,description)
Add help message for *command*

addHelpGroup(name) Adds a help group to the shell

addMode(name,prefix,errorRoutine,help,prefix,prefixHelp)
Add a command mode

initHelp(name,short,description,help)
Initialize shell help

setHistory(filename) Sets the location of the command history file

setOptionalUndef(bool) If true, command arguments default to undef

CGI Interface (requires "cs_cgi.pl")

addCGI(name,url,user,password,wd,[alias)
Creates a CGI connection

cgiConnect(name,[password])
Connects to *name* with *password*

cgiExecute(command,silent)
Executes *command* on remote server

cgiGetFile(source,[destination],[chmod],[replace],[gzip],[silent])
Copies the *source* from the server

cgiPutFile(source,[destination],[chmod],[replace],[gzip],[silent])
Copies the local file *source* to the server

Data Management

loadData(path,containers) Load stored data into one or more containers (array, hash or scalar); the quantity and data types of containers must match the call to storeData

loadNamedData(path,[container,…])
Loads named data into one or more containers

storeData(path,containers) Store data from one or more containers in a file

storeNamedData(path,name,container,…)
Store named data from one or more containers in a file

Data Processing

compare(\@A,\@B) Compare two arrays

compare(\%A,\%B) Compare two hashes

copyArray(\@array) Creates a safe copy of *array*

copyHash(\%HASH)Creates a safe copy of *HASH*

csv($CSV) Returns a hash reference for the next line of data

csvInit(path) or csvInit(\@lines)
Initializes CSV processing for *path* or *lines*.

exportData(fh,label,data) Exports *data* to an open file

exportStorage(storage,filename)Exports *storage* to *filename*

exportToFile(filename,label,data) Exports *data* to *filename*

importData(fh) Imports data from an open file

importFromFile(filename) Imports data from *filename*

makeCopy(\@A,\@B) Makes safe copy of @B in @A

makeCopy(\%A,\%B) Makes safe copy of %B in %A

Database Routines (requires "cs_db.pl")

dbDelete(table,[where]) Construtcs a DELETE statement

dbDo(sql) Processes SQL statement

dbFetch1Hash(sql) Processes SQL and returns a hash for first result

dbFetchArray(sql) Processes SQL and returns an array of results

dbFetchHash(sql) Processes SQL and returns array of hashes of results

dbInsertRow(table,\%HASH)
Inserts a row in *table* using values of *HASH*

dbIsOpen(name) Returns true if database *name* is open

dbIsRunning() Returns true if MySQL server is running

dbOpen(database,[user],[password])
Opens the database with permissions for *user*

dbPseudoField(table,field,key,type,default,null)
Defines a pseudo field added to hashes for *table*

dbQuote(value) Quotes *value* to be used in SQL statement
dbSelect(fields,table,where,order)
Constructs a SELECT statement
dbSelectHash(table,\%HASH) Constructs SQL to retrieve from *table*
dbShowQueryResults([table],result) Prints the values of all rows in result
dbTableHash(table) Returnsa row in table
dbUpdateHashes(table,key,\%OLD,\%NEW)
Updates a row in *table* with changes from *NEW*

Date/Time Processing

fmtDate(fmt,tm,gmt) Formats the number of non-leap seconds since epoch
timeDiff(a,b) Describes elapsed time between two times

Date/Time Formatting

am	either "am" or "pm"	m	single digit month
AM	either "AM" or "PM"	mm	two digit month
d	single digit day	mmm	short month name: Jan
dd	two digit day	mmmm	long month name: January
ddd	short weekday: Mon	n	single digit minute
dddd	long weekday: Monday	nn	two digit minute
h	onw digit 24-hour clock hour	s	single digit second
hh	two digit 24-hour clock hour	ss	two digit second
hhh	one digit 12-hour clock hour	yy	two digit year

Debugging

addDebugFlag(flag,value) Adds a named debug *flag*
array(\@array) Prints contents of *array* truncated to width of terminal.
debug(text,…) Prints debugging *text* to terminal
debugColor(color,source) Sets debugging *color* for *source* file
debugger(evaluator) Invokes the debugger
dumpVar(var,descend) Prints contents of *var*
first(%HASH) Prints contents of first key/value pair of *HASH*
getDebugFlag(flag) Returns the current setting for a named debug *flag*
getMainMode() Return the name of the main mode
resp($RESP) Prints contents of HTTP response
setDebugFlag(flag,value) Sets the named debug *flag* to *value*

Error Messages

abort(text) Print *text* (in red) and exit the program.
complain(text) Print *text* (in red)
complainTrace(text) Print *text* (in red) and a stack trace
unknownCommand Prints "What?" (in red)

File/Directory Processing

doDir(dir,walk,routine)
doDir(dir,walk,[rules])Traverse a directory and take designated actions on appropriate objects.
getFileList([directory],match)Returns an array of files
getHome() Returns the user's home directory
getPath(relPath) Returns the full path for a relative path
getWD() Returns the current working directory
pathFull(filename) Returns fullly qualified "/path/to/filename"
pathToSystem(filename)Returns *filename* with escaped characters

Formatted Output

chopTable(rows) Formats and displays a table (see "Formatted Tables" below); output lines are chopped at the terminal width.
comma(value) Returns comma notated value
pagedFile(filename) Prints contents of file one page at a time
pagedOutput(text) Prints *text* to screen one page at a time
pagedTable(table) Formats *table* and prints results one page at a time
showTable(rows) Formats and prints a *table*

Formatted Tables

\a%%5.2%center%chop:7 %comma%right%wrap:45
\f<full line text>
<field>\t<field>\t…

HTML Content (requires "cs_network.pl")

html2text(html) Returns *html* converted to human-readable text
htmlGetSelected(tree,name) Returns text for selected option in *tree*
htmlParse(content) Returns a parsed tree for the *content*
htmlParseFile(filename) Returns a parsed tree for contents of *filename*
htmlTag(content,tag)Searches *content* for matching *tag*
htmlTagClass(content,tag,class)Searchs *content* for matching *tag* with *class*
htmlTagId(content,tag,id) Searches *content* for matching *tag* with *ID*

HTTP Requests (requires "cs_network.pl")

httpAddHeader(name,value) Adds a header to next request
httpGet(url) Sends GET request for *url*
httpHead(url) Sends HEAD requet for *url*
httpPost(url,content) Posts *content* to *url*
httpPostContent(content) Formats name/value pairs
httpSetCookie(key,value,path,domain,port,secure,maxage)
Adds a cookie to the cookie jar for future requests
relativeURL(base,[parent],url) Returns *url* relative to *base* and *parent*

Information

codeToFile(code) Returns defining file for *code*
codeToLocation(code) Returns "line # of file" for *code*
codeToName(code) Returns the name of *code*
collectCode(\%HASH,[ref,[prefix]])
Collects information about code from symbol table
getOS() Returns a string indicating the operating system; one of cygwin, macosx, unix, or windows
getTermHeight() Returns the height of the user's terminal
getTermWidth() Returns the width of the user's terminal
toc(path,[len],[width]) Returns TOC for a source file (or collection)

Library Management

findLibrary(name) Returns an array of paths to a system library (or undef)
loadLibrary(name) Loads a Perl library

Numeric Processing

max(a,b,…) Returns the largest of a group of numbers
smin(a,b,…) Returns the smallest of a group of numbers

String Processing

displayLength(text) Returns the number of chacters that will be displayed
safeString(text) Returns safe copy of *text* (no wide characters)
safeSubstr(text,pos,n) Returns a substring *n* characters long starting at *pos*
singPlural(n,singular,plural) Returns the singular form if *n* is 1; otherwise returns the plural form.
sortAlpha(array) Returns *array* sorted alphabetically
sortAlphaIC(array) Returns *array* sorted alphabetically ignoring case
testString(text) Returns 1 bit: ASCII, 2 bit: UTF-8, 4 bit: wide
trueLength(text) Returns number of UTF-8 characters in *text*
wrapLine(text) Wraps a line of *text* so that it will fit on the terminal
wrapText(text,width) Wraps a line of *text* so that it will fit in the specified width

User Input

getNumber(prompt,low,high) Prompts the user for input and returns the result, which must be a number between *low* and *high* inclusive.
getNumberList(prompt,low,high) Prompts the user for input and returns the result(s), list of numbers between *low* and *high*
getReply(prompt,allowed,password) Prompts the user for input
getReplyInt() Returns a boolean indicating if the user interrupted the most recent call to getReply
yesNo(prompt) Prompts the user for a yes/no response; returns boolean