

Introduction

Developing and running Tcl scripts to control the Altera® Quartus® II software allows you to perform a wide range of functions, such as compiling a design or writing procedures to automate common tasks.

You can use Tcl scripts to manage a Quartus II project, make assignments, define design constraints, make device assignments, compile your design, perform timing analysis, and access reports. Tcl scripts also facilitate project or assignment migration. For example, when designing in different projects with the same prototype or development board, you can automate reassignment of pin locations in each new project. The Quartus II software can also generate a Tcl script based on all the current assignments in the project, which aids in switching assignments to another project.

The Quartus II software Tcl commands follow the EDA industry Tcl application programming interface (API) standards for command-line options. This simplifies learning and using Tcl commands. If you encounter an error with a command argument, the Tcl interpreter includes help information showing correct usage.

This chapter includes sample Tcl scripts for automating the Quartus II software. You can modify these example scripts for use with your own designs. You can find more Tcl scripts in the Design Examples section of the Support area on the Altera website.

This chapter includes the following topics:

- “Quartus II Tcl Packages” on page 3-2
- “Quartus II Tcl API Help” on page 3-3
- “Command-Line Options: -t, -s, and --tcl_eval” on page 3-5
- “End-to-End Design Flows” on page 3-7
- “Creating Projects and Making Assignments” on page 3-7
- “Compiling Designs” on page 3-8
- “Reporting” on page 3-9
- “Timing Analysis” on page 3-10
- “Automating Script Execution” on page 3-10
- “Other Scripting Features” on page 3-13
- “The Quartus II Tcl Shell in Interactive Mode” on page 3-17

- “The tclsh Shell” on page 3–18
- “Tcl Scripting Basics” on page 3–18

Tool Command Language

Tcl (pronounced “tickle”) stands for Tool Command Language, a popular scripting language that is similar to many shell scripting and high-level programming languages. It provides support for control structures, variables, network socket access, and APIs. Tcl is the EDA industry-standard scripting language used by Synopsys, Mentor Graphics®, and Altera software. It allows you to create custom commands and works seamlessly across most development platforms. For a list of recommended literature on Tcl, refer to “External References” on page 3–25.

You can create your own procedures by writing scripts containing basic Tcl commands and Quartus II API functions. You can then automate your design flow, run the Quartus II software in batch mode, or execute the individual Tcl commands interactively in the Quartus II Tcl interactive shell.

If you are unfamiliar with Tcl scripting, or are a Tcl beginner, refer to “Tcl Scripting Basics” on page 3–18 for an introduction to Tcl scripting.

The Quartus II software supports Tcl/Tk version 8.5, supplied by the Tcl DeveloperXchange at tcl.activestate.com.

Quartus II Tcl Packages

The Quartus II Tcl commands are grouped in packages by function. Table 3–1 describes each Tcl package.

Table 3–1. Tcl Packages (Part 1 of 2)

Package Name	Package Description
backannotate	Back annotate assignments
chip_planner	Identify and modify resource usage and routing with the Chip Editor
database_manager	Manage version-compatible database files
device	Get device and family information from the device database
flow	Compile a project, run command-line executables and other common flows
incremental compilation	Manipulate design partitions and LogicLock regions, and settings related to incremental compilation
insystem_memory_edit	Read and edit memory contents in Altera devices
insystem_source_probe	Interact with the In-System Sources and Probes tool in an Altera device
jtag	Control the JTAG chain
logic_analyzer_interface	Query and modify the logic analyzer interface output pin state
misc	Perform miscellaneous tasks such as enabling natural bus naming, package loading, and message posting
project	Create and manage projects and revisions, make any project assignments including timing assignments
rapid_recompile	Manipulate Quartus II Rapid Recompile features
report	Get information from report tables, create custom reports

Table 3–1. Tcl Packages (Part 2 of 2)

Package Name	Package Description
rtl	Traversing and querying the RTL netlist of your design
sdcc	Specifies constraints and exceptions to the TimeQuest Timing Analyzer
sdcc_ext	Altera-specific SDC commands
simulator	Configure and perform simulations
sta	Contains the set of Tcl functions for obtaining advanced information from the Quartus II TimeQuest Timing Analyzer
stp	Run the SignalTap® II Logic Analyzer

By default, only the minimum number of packages is loaded automatically with each Quartus II executable. This keeps the memory requirement for each executable as low as possible. Because the minimum number of packages is automatically loaded, you must load other packages before you can run commands in those packages.

Because different packages are available in different executables, you must run your scripts with executables that include the packages you use in the scripts. For example, if you use commands in the **sdcc_ext** package, you must use the **quartus_sta** executable to run the script because the **quartus_sta** executable is the only one with support for the **sdcc_ext** package.

The following command prints lists of the packages loaded or available to load for an executable, to the console:

```
<executable name> --tcl_eval help ↵
```

For example, type the following command to list the packages loaded or available to load by the **quartus_fit** executable:

```
quartus_fit --tcl_eval help ↵
```

Loading Packages

To load a Quartus II Tcl package, use the `load_package` command as follows:

```
load_package [-version <version number>] <package name>
```

This command is similar to the `package require` Tcl command (described in [Table 3–2 on page 3–4](#)), but you can easily alternate between different versions of a Quartus II Tcl package with the `load_package` command because of the `-version` option.



For additional information about these and other Quartus II command-line executables, refer to the *Command-Line Scripting* chapter in volume 2 of the *Quartus II Handbook*.

Quartus II Tcl API Help

Access the Quartus II Tcl API Help reference by typing the following command at a system command prompt:

```
quartus_sh --qhelp ↵
```

This command runs the Quartus II Command-Line and Tcl API help browser, which documents all commands and options in the Quartus II Tcl API.

Quartus II Tcl help allows easy access to information about the Quartus II Tcl commands. To access the help information, type `help` at a Tcl prompt, as shown in [Example 3-1](#).

Example 3-1. Help Output

```
tcl> help
-----

-----
Available Quartus II Tcl Packages:
-----

Loaded                                Not Loaded
-----
::quartus::misc                      ::quartus::device
::quartus::old_api                   ::quartus::backannotate
::quartus::project                   ::quartus::flow
::quartus::timing_assignment          ::quartus::logiclock
::quartus::timing_report              ::quartus::report

* Type "help -tcl"
to get an overview on Quartus II Tcl usages.
```

[Table 3-2](#) summarizes the help options available in the Tcl environment.

Table 3-2. Help Options Available in the Quartus II Tcl Environment (Part 1 of 2)

Help Command	Description
<code>help</code>	To view a list of available Quartus II Tcl packages, loaded and not loaded.
<code>help -tcl</code>	To view a list of commands used to load Tcl packages and access command-line help.
<code>help -pkg <package_name></code> <code>[-version <version number>]</code>	<p>To view help for a specified Quartus II package that includes the list of available Tcl commands. For convenience, you can omit the <code>::quartus::</code> package prefix, and type <code>help -pkg <package name></code> ↵.</p> <p>If you do not specify the <code>-version</code> option, help for the currently loaded package is displayed by default. If the package for which you want help is not loaded, help for the latest version of the package is displayed by default.</p> <p>Examples:</p> <pre>help -pkg ::quartus::project ↵ help -pkg project ↵ help -pkg project -version 1.0 ↵</pre>
<code><command_name> -h</code> or <code><command_name> -help</code>	<p>To view short help for a Quartus II Tcl command for which the package is loaded.</p> <p>Examples:</p> <pre>project_open -h ↵ project_open -help ↵</pre>

Table 3–2. Help Options Available in the Quartus II Tcl Environment (Part 2 of 2)

Help Command	Description
<pre>package require ::quartus::<package [<version>]<="" name>="" pre=""> </package></pre>	<p>To load a Quartus II Tcl package with the specified version. If <i><version></i> is not specified, the latest version of the package is loaded by default.</p> <p>Example:</p> <pre>package require ::quartus::project 1.0 ↵</pre> <p>This command is similar to the <code>load_package</code> command.</p> <p>The advantage of the <code>load_package</code> command is that you can alternate freely between different versions of the same package.</p> <p>Type <code>load_package <package name> [-version <version number>]</code> ↵ to load a Quartus II Tcl package with the specified version. If the <code>-version</code> option is not specified, the latest version of the package is loaded by default.</p> <p>Example:</p> <pre>load_package ::quartus::project -version 1.0 ↵</pre>
<pre>help -cmd <command_name> [-version <version>] or <command_name> -long_help</pre>	<p>To view complete help text for a Quartus II Tcl command.</p> <p>If you do not specify the <code>-version</code> option, help for the command in the currently loaded package version is displayed by default.</p> <p>If the package version for which you want help is not loaded, help for the latest version of the package is displayed by default.</p> <p>Examples:</p> <pre>project_open -long_help ↵ help -cmd project_open ↵ help -cmd project_open -version 1.0 ↵</pre>
<code>help -examples</code>	To view examples of Quartus II Tcl usage.
<code>help -quartus</code>	To view help on the predefined global Tcl array that contains project information and information about the Quartus II executable that is currently running.
<code>quartus_sh --qhelp</code>	<p>To launch the Tk viewer for Quartus II command-line help and display help for the command-line executables and Tcl API packages.</p> <p>For more information about this utility, refer to the Command-Line Scripting chapter in volume 2 of the <i>Quartus II Handbook</i>.</p>

❓ The Tcl API help is also available in Quartus II online help. Search for the command or package name to find details about that command or package.

Command-Line Options: -t, -s, and --tcl_eval

Table 3–3 lists three command-line options you can use with executables that support Tcl.

Table 3–3. Command-Line Options Supporting Tcl Scripting (Part 1 of 2)

Command-Line Option	Description
<code>--script=<script file> [<script args>]</code>	Run the specified Tcl script with optional arguments.
<code>-t <script file> [<script args>]</code>	Run the specified Tcl script with optional arguments. The <code>-t</code> option is the short form of the <code>--script</code> option.
<code>--shell</code>	Open the executable in the interactive Tcl shell mode.

Table 3-3. Command-Line Options Supporting Tcl Scripting (Part 2 of 2)

Command-Line Option	Description
-s	Open the executable in the interactive Tcl shell mode. The -s option is the short form of the --shell option.
--tcl_eval <tcl command>	Evaluate the remaining command-line arguments as Tcl commands. For example, the following command displays help for the project package: quartus_sh --tcl_eval help -pkg project

Run a Tcl Script

Running an executable with the -t option runs the specified Tcl script. You can also specify arguments to the script. Access the arguments through the argv variable, or use a package such as **cmdline**, which supports arguments of the following form:

```
-<argument name> <argument value>
```

The **cmdline** package is included in the <Quartus II directory>/common/tcl/packages directory.

For example, to run a script called **myscript.tcl** with one argument, Stratix, type the following command at a system command prompt:

```
quartus_sh -t myscript.tcl Stratix ↵
```

Refer to [“Accessing Command-Line Arguments” on page 3-15](#) for more information.

Interactive Shell Mode

Running an executable with the -s option starts an interactive Tcl shell. For example, to open the Quartus II TimeQuest Timing Analyzer executable in interactive shell mode, type the following command:

```
quartus_sta -s ↵
```

Commands you type in the Tcl shell are interpreted when you click **Enter**. You can run a Tcl script in the interactive shell with the following command:

```
source <script name> ↵
```

If a command is not recognized by the shell, it is assumed to be an external command and executed with the exec command.

Evaluate as Tcl

Running an executable with the --tcl_eval option causes the executable to immediately evaluate the remaining command-line arguments as Tcl commands. This can be useful if you want to run simple Tcl commands from other scripting languages.

For example, the following command runs the Tcl command that prints out the commands available in the project package.

```
quartus_sh --tcl_eval help -pkg project ↵
```

The Quartus II Tcl Console Window

You can run Tcl commands directly in the Quartus II Tcl Console window. On the View menu, click **Utility Windows**. By default, the Tcl Console window is docked in the bottom-right corner of the Quartus II GUI. All Tcl commands typed in the Tcl Console are interpreted by the Quartus II Tcl shell.



Some shell commands such as `cd`, `ls`, and others can be run in the Tcl Console window, with the Tcl `exec` command. However, for best results, run shell commands and Quartus II executables from a system command prompt outside of the Quartus II software GUI.

Tcl messages appear in the **System** tab (Messages window). Errors and messages written to `stdout` and `stderr` also are shown in the Quartus II Tcl Console window.

End-to-End Design Flows

You can use Tcl scripts to control all aspects of the design flow, including controlling other software, when the other software also includes a scripting interface.

Typically, EDA tools include their own script interpreters that extend core language functionality with tool-specific commands. For example, the Quartus II Tcl interpreter supports all core Tcl commands, and adds numerous commands specific to the Quartus II software. You can include commands in one Tcl script to run another script, which allows you to combine or chain together scripts to control different tools. Because scripts for different tools must be executed with different Tcl interpreters, it is difficult to pass information between the scripts unless one script writes information into a file and another script reads it.

Within the Quartus II software, you can perform many different operations in a design flow (such as synthesis, fitting, and timing analysis) from a single script, making it easy to maintain global state information and pass data between the operations. However, there are some limitations on the operations you can perform in a single script due to the various packages supported by each executable.

There are no limitations on running flows from any executable. Flows include operations found in the Start section of the Processing menu in the Quartus II GUI, and are also documented as options for the `execute_flow` Tcl command. If you can make settings in the Quartus II software and run a flow to get your desired result, you can make the same settings and run the same flow in a Tcl script.

Creating Projects and Making Assignments

You can easily create a script that makes all the assignments for an existing project, and then use the script at any time to restore your project settings to a known state. From the Project menu, click **Generate Tcl File for Project** to automatically generate a `.tcl` file with all of your assignments. You can source this file to recreate your project, and you can edit the file to add other commands, such as compiling the design. The file is a good starting point to learn about project management commands and assignment commands.

- Refer to “Interactive Shell Mode” on page 3-6 for information about sourcing a script. Scripting information for all Quartus II project settings and assignments is located in the *QSF Reference Manual*. Refer to the *Constraining Designs* chapter in volume 2 of the Quartus II Handbook for more information on making assignments.

Example 3-2 shows how to create a project, make assignments, and compile the project. It uses the **fir_filter** tutorial design files in the **qdesigns** installation directory. Run this script in the **fir_filter** directory, with the **quartus_sh** executable.

Example 3-2. Create and Compile a Project

```
load_package flow

# Create the project and overwrite any settings
# files that exist
project_new fir_filter -revision filtref -overwrite
# Set the device, the name of the top-level BDF,
# and the name of the top level entity
set_global_assignment -name FAMILY Cyclone
set_global_assignment -name DEVICE EP1C6F256C6
set_global_assignment -name BDF_FILE filtref.bdf
set_global_assignment -name TOP_LEVEL_ENTITY filtref
# Add other pin assignments here
set_location_assignment -to clk Pin_G1
# compile the project
execute_flow -compile
project_close
```

- The assignments created or modified while a project is open are not committed to the Quartus II Settings File (**.qsf**) unless you explicitly call **export_assignments** or **project_close** (unless **-dont_export_assignments** is specified). In some cases, such as when running **execute_flow**, the Quartus II software automatically commits the changes.

- For information about scripted design flows for HardCopy II designs, refer to the *Script-Based Design for HardCopy II Devices* chapter of the *HardCopy Handbook*. A separate chapter in the *HardCopy Handbook* called *Timing Constraints for HardCopy II Devices* also contains information about script-based design for HardCopy II devices, with an emphasis on timing constraints.

Compiling Designs

You can run the Quartus II command-line executables from Tcl scripts. Use the included **flow** package to run various Quartus II compilation flows, or run each executable directly.

The flow Package

The **flow** package includes two commands for running Quartus II command-line executables, either individually or together in standard compilation sequence. The **execute_module** command allows you to run an individual Quartus II command-line executable. The **execute_flow** command allows you to run some or all of the executables in commonly-used combinations. Use the **flow** package instead of system calls to run Quartus II executables from scripts or from the Quartus II Tcl Console.

Compile All Revisions

You can use a simple Tcl script to compile all revisions in your project. Save the script shown in [Example 3-3](#) in a file called **compile_revisions.tcl** and type the following to run it:

```
quartus_sh -t compile_revisions.tcl <project name> ↵
```

Example 3-3. Compile All Revisions

```
load_package flow
project_open [lindex $quartus(args) 0]
set original_revision [get_current_revision]
foreach revision [get_project_revisions] {
    set_current_revision $revision
    execute flow -compile
}
set_current_revision $original_revision
project_close
```

Reporting

It is sometimes necessary to extract information from the Compilation Report to evaluate results. The Quartus II Tcl API provides easy access to report data so you do not have to write scripts to parse the text report files.

If you know the exact cell or cells you want to access, use the `get_report_panel_data` command and specify the row and column names (or *x* and *y* coordinates) and the name of the appropriate report panel. You can often search for data in a report panel. To do this, use a loop that reads the report one row at a time with the `get_report_panel_row` command.

Column headings in report panels are in row 0. If you use a loop that reads the report one row at a time, you can start with row 1 to skip the row with column headings. The `get_number_of_rows` command returns the number of rows in the report panel, including the column heading row. Because the number of rows includes the column heading row, continue your loop as long as the loop index is less than the number of rows.

Report panels are hierarchically arranged and each level of hierarchy is denoted by the string “||” in the panel name. For example, the name of the Fitter Settings report panel is `Fitter||Fitter Settings` because it is in the Fitter folder. Panels at the highest hierarchy level do not use the “||” string. For example, the Flow Settings report panel is named `Flow Settings`.

The code in [Example 3-4](#) prints a list of all report panel names in your project. You can run this code with any executable that includes support for the report package.

Example 3-4. Print All Report Panel Names

```
load_package report
project_open myproject
load_report
set panel_names [get_report_panel_names]
foreach panel_name $panel_names {
    post_message "$panel_name"
}
```

Viewing Report Data in Excel

The Microsoft Excel software is sometimes used to view or manipulate timing analysis results. You can create a Comma Separated Value (.csv) file from any Quartus II report to open with Excel. [Example 3-5](#) shows a simple way to create a .csv file with data from the Fitter panel in a report. You could modify the script to use command-line arguments to pass in the name of the project, report panel, and output file to use. You can run this script example with any executable that supports the report package.

Example 3-5. Create .csv Files from Reports

```
load_package report
project_open my-project

load_report

# This is the name of the report panel to save as a CSV file
set panel_name "Fitter|Fitter Settings"
set csv_file "output.csv"

set fh [open $csv_file w]
set num_rows [get_number_of_rows -name $panel_name]

# Go through all the rows in the report file, including the
# row with headings, and write out the comma-separated data
for { set i 0 } { $i < $num_rows } { incr i } {
    set row_data [get_report_panel_row -name $panel_name \
        -row $i]
    puts $fh [join $row_data ","]
}

close $fh
unload_report
```

Timing Analysis

The Quartus II TimeQuest Timing Analyzer includes support for industry-standard SDC commands in the **sdc** package. The Quartus II software also includes comprehensive Tcl APIs and SDC extensions for the TimeQuest Timing Analyzer in the **sta**, and **sdc_ext** packages.



Refer to the [Quartus II TimeQuest Timing Analyzer](#) chapter in volume 3 of the *Quartus II Handbook* for detailed information about how to perform timing analysis with the Quartus II TimeQuest Timing Analyzer.

Automating Script Execution

You can configure scripts to run automatically at various points during compilation. Use this capability to automatically run scripts that perform custom reporting, make specific assignments, and perform many other tasks.

The following three global assignments control when a script is run automatically:

- **PRE_FLOW_SCRIPT_FILE** —before a flow starts
- **POST_MODULE_SCRIPT_FILE** —after a module finishes

- POST_FLOW_SCRIPT_FILE —after a flow finishes

A module is another term for a Quartus II executable that performs one step in a flow. For example, two modules are Analysis and Synthesis (**quartus_map**), and timing analysis (**quartus_sta**).

A flow is a series of modules that the Quartus II software runs with predefined options. For example, compiling a design is a flow that typically consists of the following steps (performed by the indicated module):

1. Analysis and synthesis (**quartus_map**)
2. Fitter (**quartus_fit**)
3. Assembler (**quartus_asm**)
4. Timing Analyzer (**quartus_sta**)

Other flows are described in the help for the `execute_flow` Tcl command. In addition, many commands in the Processing menu of the Quartus II GUI correspond to this design flow.

To make an assignment automatically run a script, add an assignment with the following form to the **.qsf** for your project:

```
set_global_assignment -name <assignment name> <executable>:<script name>
```

The Quartus II software runs the scripts as shown in [Example 3-6](#).

Example 3-6.

```
<executable> -t <script name> <flow or module name> <project name> <revision name>
```

The first argument passed in the `argv` variable (or `quartus(args)` variable) is the name of the flow or module being executed, depending on the assignment you use. The second argument is the name of the project and the third argument is the name of the revision.

When you use the `POST_MODULE_SCRIPT_FILE` assignment, the specified script is automatically run after every executable in a flow. You can use a string comparison with the module name (the first argument passed in to the script) to isolate script processing to certain modules.

Execution Example

[Example 3-7](#) illustrates how automatic script execution works in a complete flow, assuming you have a project called **top** with a current revision called **rev_1**, and you have the following assignments in the **.qsf** for your project.

Example 3-7.

```
set_global_assignment -name PRE_FLOW_SCRIPT_FILE quartus_sh:first.tcl
set_global_assignment -name POST_MODULE_SCRIPT_FILE quartus_sh:next.tcl
set_global_assignment -name POST_FLOW_SCRIPT_FILE quartus_sh:last.tcl
```

When you compile your project, the `PRE_FLOW_SCRIPT_FILE` assignment causes the following command to be run before compilation begins:

```
quartus_sh -t first.tcl compile top rev_1
```

Next, the Quartus II software starts compilation with analysis and synthesis, performed by the **quartus_map** executable. After the analysis and synthesis finishes, the POST_MODULE_SCRIPT_FILE assignment causes the following command to run:

```
quartus_sh -t next.tcl quartus_map top rev_1
```

Then, the Quartus II software continues compilation with the Fitter, performed by the **quartus_fit** executable. After the Fitter finishes, the POST_MODULE_SCRIPT_FILE assignment runs the following command:

```
quartus_sh -t next.tcl quartus_fit top rev_1
```

Corresponding commands are run after the other stages of the compilation. When the compilation is over, the POST_FLOW_SCRIPT_FILE assignment runs the following command:

```
quartus_sh -t last.tcl compile top rev_1
```

Controlling Processing

The POST_MODULE_SCRIPT_FILE assignment causes a script to run after every module. Because the same script is run after every module, you might have to include some conditional statements that restrict processing in your script to certain modules.

For example, if you want a script to run only after timing analysis, use a conditional test like the one shown in [Example 3-8](#). It checks the flow or module name passed as the first argument to the script and executes code when the module is **quartus_sta**.

Example 3-8. Restrict Processing to a Single Module

```
set module [lindex $quartus(args) 0]

if [string match "quartus_sta" $module] {

    # Include commands here that are run
    # after timing analysis
    # Use the post-message command to display
    # messages
    post_message "Running after timing analysis"

}
```

Displaying Messages

Because of the way the Quartus II software runs the scripts automatically, you must use the `post_message` command to display messages, instead of the `puts` command. This requirement applies only to scripts that are run by the three assignments listed in [“Automating Script Execution” on page 3-10](#).



Refer to [“The post_message Command” on page 3-14](#) for more information about this command.

Other Scripting Features

The Quartus II Tcl API includes other general-purpose commands and features described in this section.

Natural Bus Naming

The Quartus II software supports natural bus naming. Natural bus naming allows you to use square brackets to specify bus indexes in HDL without including escape characters to prevent Tcl from interpreting the square brackets as containing commands. For example, one signal in a bus named `address` can be identified as `address[0]` instead of `address\[0\]`. You can take advantage of natural bus naming when making assignments, as in [Example 3-9](#).

Example 3-9. Natural Bus Naming

```
set_location_assignment -to address[10] Pin_M20
```

The Quartus II software defaults to natural bus naming. You can turn off natural bus naming with the `disable_natural_bus_naming` command. For more information about natural bus naming, type the following at a Quartus II Tcl prompt:

```
enable_natural_bus_naming -h ←
```

Short Option Names

You can use short versions of command options, as long as they are unambiguous. For example, the `project_open` command supports two options: `-current_revision` and `-revision`. You can use any of the following abbreviations of the `-revision` option: `-r`, `-re`, `-rev`, `-revi`, `-revis`, and `-revisio`. You can use an option as short as `-r` because in the case of the `project_open` command no other option starts with the letter `r`. However, the `report_timing` command includes the options `-recovery` and `-removal`. You cannot use `-r` or `-re` to shorten either of those options, because the abbreviation would not be unique to only one option.

Collection Commands

Some Quartus II Tcl functions return very large sets of data that would be inefficient as Tcl lists. These data structures are referred to as collections. The Quartus II Tcl API uses a collection ID to access the collection. There are two Quartus II Tcl commands for working with collections, `foreach_in_collection` and `get_collection_size`. Use the `set` command to assign a collection ID to a variable.

- ❓ For information about which Quartus II Tcl commands return collection IDs, refer to [foreach_in_collection](#) in Quartus II Help.

The foreach_in_collection Command

The `foreach_in_collection` command is similar to the `foreach` Tcl command. Use it to iterate through all elements in a collection. [Example 3-10](#) prints all instance assignments in an open project.

Example 3-10. Collection Commands

```
set all_instance_assignments [get_all_instance_assignments -name *]
foreach_in_collection asgn $all_instance_assignments {
    # Information about each assignment is
    # returned in a list. For information
    # about the list elements, refer to Help
    # for the get-all-instance-assignments command.
    set to [lindex $asgn 2]
    set name [lindex $asgn 3]
    set value [lindex $asgn 4]
    puts "Assignment to $to: $name = $value"
}
```

The get_collection_size Command

Use the `get_collection_size` command to get the number of elements in a collection. [Example 3-11](#) prints the number of global assignments in an open project.

Example 3-11. get_collection_size Command

```
set all_global_assignments [get_all_global_assignments -name *]
set num_global_assignments [get_collection_size $all_global_assignments]
puts "There are $num_global_assignments global assignments in your project"
```

The post_message Command

To print messages that are formatted like Quartus II software messages, use the `post_message` command. Messages printed by the `post_message` command appear in the **System** tab of the Messages window in the Quartus II GUI, and are written to standard at when scripts are run. Arguments for the `post_message` command include an optional message type and a required message string.

The message type can be one of the following:

- info (default)
- extra_info
- warning
- critical_warning
- error

If you do not specify a type, Quartus II software defaults to `info`.

With the Quartus II software in Windows, you can color code messages displayed at the system command prompt with the `post_message` command. Add the following line to your **quartus2.ini** file:

```
DISPLAY_COMMAND_LINE_MESSAGES_IN_COLOR = on
```

[Example 3-12](#) shows how to use the `post_message` command.

Example 3-12. `post_message` command

```
post_message -type warning "Design has gated clocks"
```

Accessing Command-Line Arguments

Many Tcl scripts are designed to accept command-line arguments, such as the name of a project or revision. The global variable `quartus(args)` is a list of the arguments typed on the command-line following the name of the Tcl script. [Example 3-13](#) shows code that prints all of the arguments in the `quartus(args)` variable.

Example 3-13. Simple Command-Line Argument Access

```
set i 0
foreach arg $quartus(args) {
    puts "The value at index $i is $arg"
    incr i
}
```

If you copy the script in the previous example to a file named `print_args.tcl`, it displays the following output when you type the command shown in [Example 3-14](#) at a command prompt.

Example 3-14. Passing Command-Line Arguments to Scripts

```
quartus_sh -t print_args.tcl my_project 100MHz ↵
The value at index 0 is my_project
The value at index 1 is 100MHz
```

The `cmdline` Package

You can use the `cmdline` package included with the Quartus II software for more robust and self-documenting command-line argument passing. The `cmdline` package supports command-line arguments with the form `-<option> <value>`.

[Example 3-15](#) uses the `cmdline` package.

Example 3-15. `cmdline` Package

```
package require cmdline
variable ::argv0 $::quartus(args)
set options {
    { "project.arg" "" "Project name" }
    { "frequency.arg" "" "Frequency" }
}
set usage "You need to specify options and values"

array set optshash [::cmdline::getoptions ::argv $options $usage]
puts "The project name is $optshash(project)"
puts "The frequency is $optshash(frequency)"
```

If you save those commands in a Tcl script called `print_cmd_args.tcl` you see the following output when you type the command shown in [Example 3-16](#) at a command prompt.

Example 3-16. Passing Command-Line Arguments for Scripts

```
quartus_sh -t print_cmd_args.tcl -project my_project -frequency 100MHz r
The project name is my_project
The frequency is 100MHz
```

Virtually all Quartus II Tcl scripts must open a project. [Example 3-17](#) opens a project, and you can optionally specify a revision name. The example checks whether the specified project exists. If it does, the example opens the current revision, or the revision you specify.

Example 3-17. Full-Featured Method to Open Projects

```
package require cmdline
variable ::argv0 $::quartus(args)
set options { \
{ "project.arg" "" "Project Name" } \
{ "revision.arg" "" "Revision Name" } \
}
array set optshash [::cmdline::getoptions ::argv0 $options]

# Ensure the project exists before trying to open it
if {[project_exists $optshash(project)]} {

    if {[string equal "" $optshash(revision)]} {

        # There is no revision name specified, so default
        # to the current revision
        project_open $optshash(project) -current_revision
    } else {

        # There is a revision name specified, so open the
        # project with that revision
        project_open $optshash(project) -revision \
            $optshash(revision)
    }
} else {
    puts "Project $optshash(project) does not exist"
    exit 1
}
# The rest of your script goes here
```

If you do not require this flexibility or error checking, you can use just the `project_open` command, as shown in [Example 3-18](#).

Example 3-18. Simple Method to Open Projects

```
set proj_name [lindex $argv 0]
project_open $proj_name
```

The quartus() Array

The scripts in the preceding examples parsed command line arguments found in `quartus(args)`. The global `quartus()` Tcl array includes other information about your project and the current Quartus II executable that might be useful to your scripts. For information on the other elements of the `quartus()` array, type the following command at a Tcl prompt:

```
help -quartus ↵
```

The Quartus II Tcl Shell in Interactive Mode

This section presents how to make project assignments and then compile the finite impulse response (FIR) filter tutorial project with the `quartus_sh` interactive shell. This example assumes that you already have the **fir_filter** tutorial design files in a project directory.

To begin, type the following at the system command prompt to run the interactive Tcl shell:

```
quartus_sh -s ↵
```

Create a new project called **fir_filter**, with a revision called **filtref** by typing the following command at a Tcl prompt:

```
project_new -revision filtref fir_filter ↵
```



If the project file and project name are the same, the Quartus II software gives the revision the same name as the project.

Because the revision named **filtref** matches the top-level file, all design files are automatically picked up from the hierarchy tree.

Next, set a global assignment for the device with the following command:

```
set_global_assignment -name family Cyclone ↵
```



To learn more about assignment names that you can use with the `-name` option, refer to Quartus II Help.



For assignment values that contain spaces, enclose the value in quotation marks.

To quickly compile a design, use the `::quartus::flow` package, which properly exports the new project assignments and compiles the design with the proper sequence of the command-line executables. First, load the package:

```
load_package flow ↵
```

It returns the following:

```
1.0
```

To perform a full compilation of the FIR filter design, use the `execute_flow` command with the `-compile` option:

```
execute_flow -compile ↵
```

This command compiles the FIR filter tutorial project, exporting the project assignments and running `quartus_map`, `quartus_fit`, `quartus_asm`, and `quartus_sta`. This sequence of events is the same as selecting **Start Compilation** from the Processing menu in the Quartus II GUI.

When you are finished with a project, close it with the `project_close` command as shown in [Example 3-19](#).

Example 3-19.

```
project_close ↵
```

To exit the interactive Tcl shell, type `exit` ↵ at a Tcl prompt.

The tclsh Shell

On the UNIX and Linux operating systems, the tclsh shell included with the Quartus II software is initialized with a minimal `PATH` environment variable. As a result, system commands might not be available within the tclsh shell because certain directories are not in the `PATH` environment variable. To include other directories in the path searched by the tclsh shell, set the `QUARTUS_INIT_PATH` environment variable before running the tclsh shell. Directories in the `QUARTUS_INIT_PATH` environment variable are searched by the tclsh shell when you execute a system command.

Tcl Scripting Basics

The core Tcl commands support variables, control structures, and procedures. Additionally, there are commands for accessing the file system and network sockets, and running other programs. You can create platform-independent graphical interfaces with the Tk widget set.

Tcl commands are executed immediately as they are typed in an interactive Tcl shell. You can also create scripts (including the examples in this chapter) in files and run them with the Quartus II executables or with the tclsh shell.

Hello World Example

The following shows the basic “Hello world” example in Tcl:

```
puts "Hello world" ↵
```

Use double quotation marks to group the words `hello` and `world` as one argument. Double quotation marks allow substitutions to occur in the group. Substitutions can be simple variable substitutions, or the result of running a nested command, described in [“Substitutions” on page 3-19](#). Use curly braces `{ }` for grouping when you want to prevent substitutions.

Variables

Assign a value to a variable with the `set` command. You do not have to declare a variable before using it. Tcl variable names are case-sensitive. [Example 3-20](#) assigns the value 1 to the variable named `a`.

Example 3-20. Assigning Variables

```
set a 1
```

To access the contents of a variable, use a dollar sign (“\$”) before the variable name. [Example 3-21](#) prints “Hello world” in a different way.

Example 3-21. Accessing Variables

```
set a Hello  
set b world  
puts "$a $b"
```

Substitutions

Tcl performs three types of substitution:

- Variable value substitution
- Nested command substitution
- Backslash substitution

Variable Value Substitution

Variable value substitution, as shown in [Example 3-21](#), refers to accessing the value stored in a variable with a dollar sign (“\$”) before the variable name.

Nested Command Substitution

Nested command substitution refers to how the Tcl interpreter evaluates Tcl code in square brackets. The Tcl interpreter evaluates nested commands, starting with the innermost nested command, and commands nested at the same level from left to right. Each nested command result is substituted in the outer command.

[Example 3-22](#) sets `a` to the length of the string `foo`.

Example 3-22. Command Substitution

```
set a [string length foo]
```

Backslash Substitution

Backslash substitution allows you to quote reserved characters in Tcl, such as dollar signs (“\$”) and braces (“[]”). You can also specify other special ASCII characters like tabs and new lines with backslash substitutions. The backslash character is the Tcl line continuation character, used when a Tcl command wraps to more than one line.

[Example 3-23](#) shows how to use the backslash character for line continuation.

Example 3-23. Backslash Substitution

```
set this_is_a_long_variable_name [string length "Hello \
world."]
```

Arithmetic

Use the `expr` command to perform arithmetic calculations. Use curly braces (“{ }”) to group the arguments of this command for greater efficiency and numeric precision.

[Example 3-24](#) sets `b` to the sum of the value in the variable `a` and the square root of 2.

Example 3-24. Arithmetic with the `expr` Command

```
set a 5
set b [expr { $a + sqrt(2) }]
```

Tcl also supports boolean operators such as `&&` (AND), `||` (OR), `!` (NOT), and comparison operators such as `<` (less than), `>` (greater than), and `==` (equal to).

Lists

A Tcl list is a series of values. Supported list operations include creating lists, appending lists, extracting list elements, computing the length of a list, sorting a list, and more. [Example 3-25](#) sets `a` to a list with three numbers in it.

Example 3-25. Creating Simple Lists

```
set a { 1 2 3 }
```

You can use the `lindex` command to extract information at a specific index in a list. Indexes are zero-based. You can use the `index end` to specify the last element in the list, or the `index end-<n>` to count from the end of the list. [Example 3-26](#) prints the second element (at index 1) in the list stored in `a`.

Example 3-26. Accessing List Elements

```
puts [lindex $a 1]
```

The `llength` command returns the length of a list. [Example 3-27](#) prints the length of the list stored in `a`.

Example 3-27. List Length

```
puts [llength $a]
```

The `lappend` command appends elements to a list. If a list does not already exist, the list you specify is created. The list variable name is not specified with a dollar sign ("`$`"). [Example 3-28](#) appends some elements to the list stored in `a`.

Example 3-28. Appending to a List

```
lappend a 4 5 6
```

Arrays

Arrays are similar to lists except that they use a string-based index. Tcl arrays are implemented as hash tables. You can create arrays by setting each element individually or with the `array set` command. To set an element with an index of `Mon` to a value of `Monday` in an array called `days`, use the following command:

```
set days(Mon) Monday
```

The `array set` command requires a list of index/value pairs. This example sets the array called `days`:

```
array set days { Sun Sunday Mon Monday Tue Tuesday \
               Wed Wednesday Thu Thursday Fri Friday Sat Saturday }
```

[Example 3-29](#) shows how to access the value for a particular index.

Example 3-29. Accessing Array Elements

```
set day_abbreviation Mon
puts $days($day_abbreviation)
```

Use the `array names` command to get a list of all the indexes in a particular array. The index values are not returned in any specified order. [Example 3-30](#) shows one way to iterate over all the values in an array.

Example 3-30. Iterating Over Arrays

```
foreach day [array names days] {
    puts "The abbreviation $day corresponds to the day \
name $days($day)"
}
```

Arrays are a very flexible way of storing information in a Tcl script and are a good way to build complex data structures.

Control Structures

Tcl supports common control structures, including if-then-else conditions and for, foreach, and while loops. The position of the curly braces as shown in the following examples ensures the control structure commands are executed efficiently and correctly. [Example 3-31](#) prints whether the value of variable a positive, negative, or zero.

Example 3-31. If-Then-Else Structure

```
if { $a > 0 } {  
    puts "The value is positive"  
}  
elseif { $a < 0 } {  
    puts "The value is negative"  
}  
else {  
    puts "The value is zero"  
}
```

[Example 3-32](#) uses a for loop to print each element in a list.

Example 3-32. For Loop

```
set a { 1 2 3 }  
for { set i 0 } { $i < [llength $a] } { incr i } {  
    puts "The list element at index $i is [lindex $a $i]"  
}
```

[Example 3-33](#) uses a foreach loop to print each element in a list.

Example 3-33. foreach Loop

```
set a { 1 2 3 }  
foreach element $a {  
    puts "The list element is $element"  
}
```

[Example 3-34](#) uses a while loop to print each element in a list.

Example 3-34. while Loop

```
set a { 1 2 3 }  
set i 0  
while { $i < [llength $a] } {  
    puts "The list element at index $i is [lindex $a $i]"  
    incr i  
}
```

You do not have to use the `expr` command in boolean expressions in control structure commands because they invoke the `expr` command automatically.

Procedures

Use the `proc` command to define a Tcl procedure (known as a subroutine or function in other scripting and programming languages). The scope of variables in a procedure is local to the procedure. If the procedure returns a value, use the `return` command to return the value from the procedure. [Example 3-35](#) defines a procedure that multiplies two numbers and returns the result.

Example 3-35. Simple Procedure

```
proc multiply { x y } {  
    set product [expr { $x * $y }]  
    return $product  
}
```

[Example 3-36](#) shows how to use the `multiply` procedure in your code. You must define a procedure before your script calls it.

Example 3-36. Using a Procedure

```
proc multiply { x y } {  
    set product [expr { $x * $y }]  
    return $product  
}  
set a 1  
set b 2  
puts [multiply $a $b]
```

Define procedures near the beginning of a script. If you want to access global variables in a procedure, use the `global` command in each procedure that uses a global variable. [Example 3-37](#) defines a procedure that prints an element in a global list of numbers, then calls the procedure.

Example 3-37. Accessing Global Variables

```
proc print_global_list_element { i } {  
    global my_data  
    puts "The list element at index $i is [lindex $my_data $i]"  
}  
set my_data { 1 2 3}  
print_global_list_element 0
```

File I/O

Tcl includes commands to read from and write to files. You must open a file before you can read from or write to it, and close it when the read and write operations are done. To open a file, use the `open` command; to close a file, use the `close` command. When you open a file, specify its name and the mode in which to open it. If you do not specify a mode, Tcl defaults to read mode. To write to a file, specify `w` for write mode as shown in [Example 3-38](#).

Example 3-38. Open a File for Writing

```
set output [open myfile.txt w]
```

Tcl supports other modes, including appending to existing files and reading from and writing to the same file.

The open command returns a file handle to use for read or write access. You can use the puts command to write to a file by specifying a filehandle, as shown in

[Example 3-39](#).

Example 3-39. Write to a File

```
set output [open myfile.txt w]
puts $output "This text is written to the file."
close $output
```

You can read a file one line at a time with the gets command. [Example 3-40](#) uses the gets command to read each line of the file and then prints it out with its line number.

Example 3-40. Read from a File

```
set input [open myfile.txt]
set line_num 1
while { [gets $input line] >= 0 } {
    # Process the line of text here
    puts "$line_num: $line"
    incr line_num
}
close $input
```

Syntax and Comments

Arguments to Tcl commands are separated by white space, and Tcl commands are terminated by a newline character or a semicolon. As shown in [“Substitutions” on page 3-19](#), you must use backslashes when a Tcl command extends more than one line.

Tcl uses the hash or pound character (#) to begin comments. The # character must begin a comment. If you prefer to include comments on the same line as a command, be sure to terminate the command with a semicolon before the # character.

[Example 3-41](#) is a valid line of code that includes a set command and a comment.

Example 3-41. Comments

```
set a 1;# Initializes a
```

Without the semicolon, it would be an invalid command because the set command would not terminate until the new line after the comment.

The Tcl interpreter counts curly braces inside comments, which can lead to errors that are difficult to track down. [Example 3-42](#) causes an error because of unbalanced curly braces.

Example 3-42. Unbalanced Braces in Comments

```
# if { $x > 0 } {
if { $y > 0 } {
    # code here
}
```

External References



For more information about Tcl, refer to the following sources:

- *Practical Programming in Tcl and Tk*, Brent B. Welch
- *Tcl and the TK Toolkit*, John Ousterhout
- *Effective Tcl/TK Programming*, Michael McLennan and Mark Harrison
- Quartus II Tcl example scripts at www.altera.com/support/examples/tcl/tcl.html
- Tcl Developer Xchange at tcl.activestate.com

Document Revision History

Table 3-4 shows the revision history for this chapter.

Table 3-4. Document Revision History

Date	Version	Changes
November 2011	11.0.1	<ul style="list-style-type: none"> ■ Template update ■ Updated supported version of Tcl in the section “Tool Command Language” on page 3-2 ■ minor editorial changes
May 2011	11.0.0	Minor updates throughout document.
December 2010	10.1.0	Template update Updated to remove tcl packages used by the Classic Timing Analyzer
July 2010	10.0.0	Minor updates throughout document.
November 2009	9.1.0	<ul style="list-style-type: none"> ■ Removed LogicLock example. ■ Added the incremental_compilation, insystem_source_probe, and rtl packages to Table 3-1 and Table 3-2. ■ Added quartus_map to table 3-2.
March 2009	9.0.0	<ul style="list-style-type: none"> ■ Removed the “EDA Tool Assignments” section ■ Added the section “Compile All Revisions” on page 3-9 ■ Added the section “Using the tclsh Shell” on page 3-20
November 2008	8.1.0	Changed to 8½” × 11” page size. No change to content.
May 2008	8.0.0	Updated references.



For previous versions of the *Quartus II Handbook*, refer to the [Quartus II Handbook Archive](#).



Take an [online survey](#) to provide feedback about this handbook chapter.

