HDL Designer Series™ Tcl Reference Manual

Software Version 2010.3

June, 2011

Contractor/manufacturer is:
Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777.
Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210
Website: www.mentor.com
SupportNet: supportnet.mentor.com/
Send Feedback on Documentation: supportnet.mentor.com/user/feedback_form.cfm

Table of Contents

List of Tables

Chapter 1
HDS Library Contents API

Introduction

This document is designed to help you use the HDS library contents API more effectively. The HDS library contents API enables Tcl scripts to:

1. Access the structure of a design
2. Navigate the libraries, files and declarations of the design
3. Examine the hierarchies produced by component instantiation.

For more information on the Tcl language refer to:

`http://www.tcl.tk/man/tcl8.5/tutorial/tcltutorial.html`

API Command Modes

The API commands can be used in one of two modes:

- **Interactive:** via the console page of the HDS log window. This mode provides a convenient way of learning about the API and debugging scripts, since the commands provide considerable feedback on how they should be used.This is described in more detail in the following sections.
- **Batch:** via the -tcl command line switch in HDS.

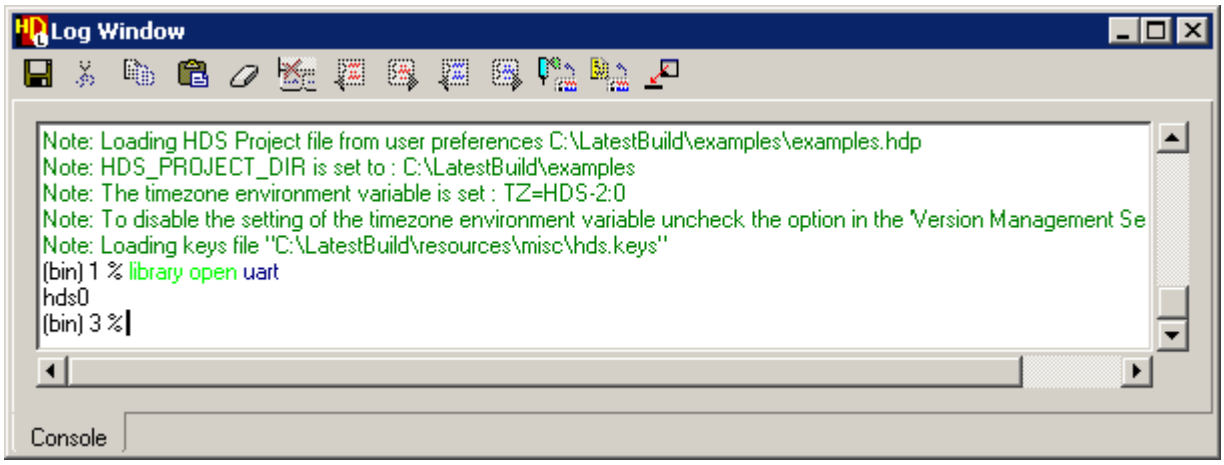
Accessing the Console

Text entered here will be executed as Tcl commands and can use the API commands described in the following sections.

To access the log window:

1. Invoke HDS to display the Design Manager window.
2. Select **Window > HDS Log Window**.

Figure 1-1. 2. Log Window



Document Conventions

In the following examples, text in bold represents commands typed into the console, text in italics represents output from HDS in response to a command.

Command Basics

HDS API commands are either dynamically created or predefined. To start using the API commands you should gain access to your design library using the **library open** command:**library open** is an example of a predefined command. HDS responds to the command by creating a handle that refers to the library object to be accessed. HDS created handles are designated by *hds* followed by a unique number.

In this example *hds0* is a handle referring to the library UART.

```
library open uart                    hds0
```

The following is an example of a dynamically created command composed of the above returned UART library handle and the files option.The command returns a set of handles referring to the UART library files. Refer to “library object commands” on page 123.

```
hds0 files                           hds1 hds2 hds3 hds4 hds5 hds6
                                     hds7 hds8 hds9 hds10 hds11 hds12
                                     hds13 hds14
```

The Tcl language supports the concept of variables. Variables are set and unset with the standard Tcl **`set`** and **`unset`** commands. To ease the use of the created handles, they are stored

in variables. In the following example the set command stores the UART handle *hds0* in variable **lib**

```
set lib [library open uart]            hds0
```

Notice the use of nested commands where the result of the second command **[library open uart]** is used as a part of the first command.

You can now use the lib variable which carries a handle to the UART library instead of the handle itself in the command.

```
$lib files                             hds1 hds2 hds3 hds4 hds5 hds6
                                       hds7 hds8 hds9 hds10 hds11 hds12
                                       hds13 hds14
```

All objects have a ‘configure’ command option, this provides access to the attributes of the object. If you use the configure command with no arguments, it shows the names and values of all the attributes.

To get a list of all object command options you can type:

```
$lib help
Error: bad option "help" should be configure or file
```

Example Scripts

This section provides and explains some sample scripts that show how an HDS design can be accessed using the Tcl API commands.

Generated HDL File Paths

Some source files of a design may be HDS graphics files. Using the Tcl API commands you can navigate to and print out the HDL text files generated from the graphics:

Steps

By copying the Tcl commands in the following table and pasting it into the HDS log window, HDS responds with a set of handles referring to the designated HDS objects and finally prints out the path to the uart_tb generated HDL file.

Table 1-1. Generated HDL Commands

Tcl Command	HDS Response
set lib [library open uart]	hds0
set tb [$lib declaration uart_tb struct]	hds1
set generatedHdlFile [[$tb file] generated]	hds43
puts [$lib configure hardHdlDir]/[$generatedHdlFile configure relativePathname]	C:/Lastbuild/32/LatestBuild/examples/uart/hdl/uart_tb_struct.vhd

Now let us see how this piece of code works.

```
set lib [library open UART]
```

A variable lib is automatically declared using the Set tcl command. The value of the variable lib is substituted with the value of the command within []

In other words the result of the library open command is given to the variable lib. Refer to "library object commands" on page 123. When a '[' appears in a command, Tcl treats everything between it and the matching ']' as a nested Tcl command. Tcl evaluates the nested API command and substitutes its result into the enclosing command in place of the bracketed text. So the above command is translated as follows:

```
set lib hds0
```

where hds0 is an HDS handle referring to the library UART

set tb [$lib declaration uart_tb struct]

Again the value of the automatically declared variable tb is substituted with the result of the second command.

When a $ appears in a command, Tcl treats the letters and digits following it as a variable name, and substitutes the value of the variable in place of the name.

So $lib carries the value hds0 mentioned in the first step which is a handle referring to the UART library. The $lib together with the declaration option and the uart_tb struct argument constitute a new command.This command returns hds1 which is a handle referring to an HDS declaration. Refer to "library object commands" on page 123.

set generatedHdlFile [[$tb file] generated]

In this step you can see that there are multiple nested Tcl API commands. The value *hds1*of the command **[$tb file]** is used to create a second dynamic command i.e. **hds1 generated**. The value of the second command substitutes the value of the variable generatedHDLFile.

puts [$lib configure hardHdlDir]/[$generatedHdlFile configure relativePathname]

Finally you can print out the path of the UART_tb generated file using the puts Tcl command. Refer to “library object commands” on page 123 and “file object commands” on page 116.

```
/home/designs/UART/hdl/uart_tb_struct.vhd
```

Reporting VHDL Packages Directly Referenced by a Library

Instead of using discrete Tcl commands, Tcl procedures will be used. Procedures in Tcl serve much the same purpose as functions in C. They may take arguments, and may return values. The basic syntax for defining a procedure is:

```
proc name argList body
```

Once a procedure is created, it is considered to be a command, just like any other built-in Tcl command. As such, it may be called using its name, followed by a value for each of its arguments. The return value from a procedure is equivalent to the result of a built-in Tcl command. Thus, command substitution can be used to substitute the return value of a procedure into another expression.

In this example the procedure showPkgs is considered a command which takes a library object as an argument and finally prints out the VHDL packages directly referenced by that library.

The API refers to entities and architectures, since these correspond to the separation of interface and implementation found both in VHDL designs and in HDS graphics files. A Verilog module corresponds to both an entity and architecture, so each module in Verilog text files becomes both an entity and an architecture in the API. This allows scripts using the API to treat VHDL text, Verilog text and HDS graphics in a consistent fashion.

All objects have a class attribute. The class attribute may have one of the following values (architecture, architectureFrame, blockFrame, caseFrame, configuration, directory, elseFrame, embeddedFrame, entity, file, forFrame, frameConfiguration, ifFrame, instance, library, machineFrame, packageBody or packageHeader).

Objects of class entity, architecture, packageHeader and packageBody support a 'packages' command that returns a list of packages used by the declaration. Refer to“Dynamically Created Commands” on page 103.

Steps

```
proc showPkgs {lib} {
   foreach file [$lib files] {
      foreach decl [$file declarations] {
         foreach pkg [$decl packages] {
           if {![info exists alreadyDone($pkg)]} {
               puts "[$pkg configure fullName]"
               set alreadyDone($pkg) 1
           }
```

```
            }
         }
      }
   }
   showPkgs [library open uart]

   ieee.std_logic_1164
   ieee.std_logic_arith
   std.TEXTIO
```

Note

You can ask a Verilog declaration for its packages, the result is always an empty list. This uniformity can be helpful when examining mixed language designs.

Reporting Verilog Include Files

File objects support an 'includes' command that provides access to the files `included by a file. Unlike most API functions, the include files are given in text form. This is because include files are not constrained to reside within HDS libraries, they can be present anywhere in the filesystem.

The following example reports all files included by the files of a library:

```
proc showIncls {lib} {
   foreach file [$lib files] {
      foreach inc [$file includes] {
         if {![info exists alreadyDone($inc)]} {
            puts $inc
            set alreadyDone($inc) 1
         }
      }
   }
}
```

Refer to “library object commands” on page 123 and “file object commands” on page 116.

Issue: our example designs have no include statements.

Note

You can ask a VHDL file for its includes, the result is always an empty list. This uniformity can be helpful when examining mixed language designs.

Simple Design Hierarchy Traversal

The following example shows a script that walks the hierarchy of a design and reports its structure:

```
proc indent {} { return [string repeat { } $::indent] }
```

The procedure indent is used as a command in the procedure walkArch. It adds indentations to the produced results. Indent has 0 arguments and uses the string repeat command to return a space repeated a number of times equal to the value of the variable $:: indent

```
proc walkArch {arch} {
   puts "[indent]ARCH [$arch configure fullName]"
   incr ::indent
   foreach inst [$arch instances] {
      puts "[indent]INST [$inst configure name]"
      walkArch [$inst child]
   }
   incr ::indent -1
   puts "[ indent]-- end [$arch configure fullName]"
}
set lib [library open uart]
set indent 0
walkArch [$lib declaration uart_tb struct]
```

Refer to "library object commands" on page 123, "instance object commands" on page 122 and "architecture object commands" on page 106.

The procedure walkArch takes a library declaration as an argument and returns all the library architectures and their instances.

```
ARCH UART.uart_tb(struct)
 INST U_0
 ARCH UART.tester(flow)
 -- end UART.tester(flow)
 INST U_1
 ARCH UART.uart_top(struct)
  INST U_1
  ARCH UART.cpu_interface(intconx)
   INST U_0
   ARCH UART.control_operation(fsm)
   -- end UART.control_operation(fsm)
  -- end UART.cpu_interface(intconx)
  INST U_2
  ARCH UART.clock_divider(flow)
  -- end UART.clock_divider(flow)
  INST U_3
  ARCH UART.address_decode(tbl)
  -- end UART.address_decode(tbl)
  INST U_4
  ARCH UART.serial_interface(struct)
   INST ser_out_mux
Error: Instance ser_out_mux is unbound
```

In this case, the command reports an error, since moduleware instances do not have a corresponding architecture.

Robust Design Hierarchy Traversal

If you want your script to trap unbound component instances and continue walking the hierarchy, you need to use the 'catch' command. The catch command evaluates a script and traps exceptional returns

```
proc indent {} { return [string repeat { } $::indent] }
proc walkArch {arch} {
   puts "[indent]ARCH [$arch configure fullName]"
   incr ::indent
   foreach inst [$arch instances] {
      if {![catch {$inst child} child]} {
         puts "[indent]INST [$inst configure name]"
         walkArch $child
      }
   }
   incr ::indent -1
   puts "[ indent]-- end [$arch configure fullName]"
}
set lib [library open uart]
set indent 0
walkArch [$lib declaration uart_tb struct]
```

Refer to “library object commands” on page 123, “instance object commands” on page 122 and “architecture object commands” on page 106.

```
ARCH UART.uart_tb(struct)
 INST U_0
  ARCH UART.tester(flow)
 -- end UART.tester(flow)
 INST U_1
  ARCH UART.uart_top(struct)
  INST U_1
   ARCH UART.cpu_interface(intconx)
   INST U_0
    ARCH UART.control_operation(fsm)
   -- end UART.control_operation(fsm)
  -- end UART.cpu_interface(intconx)
  INST U_2
   ARCH UART.clock_divider(flow)
  -- end UART.clock_divider(flow)
  INST U_3
   ARCH UART.address_decode(tbl)
  -- end UART.address_decode(tbl)
  INST U_4
   ARCH UART.serial_interface(struct)
   INST U_0
    ARCH UART.xmit_rcv_control(fsm)
   -- end UART.xmit_rcv_control(fsm)
   INST U_1
    ARCH UART.status_registers(spec)
   -- end UART.status_registers(spec)
  -- end UART.serial_interface(struct)
 -- end UART.uart_top(struct)
-- end UART.uart_tb(struct)
```

Design Hierarchy and Frames

The API uses the term 'frame' to refer to a number of constructs that add hierarchy within an architecture:

- FOR frames, IF frames, and BLOCK frames in a VHDL block diagram
- FOR frames, IF frames and ELSE frames in a Verilog block diagram
- FOR GENERATE, IF GENERATE and BLOCK statements in a VHDL text file
- FOR GENERATE statements in a Verilog text file
- Concurrent and hierarchical states in a state machine or algorithmic state machine (ASM)
- Concurrent flowcharts and hierarchical actions in a flow chart

With a slightly more complex script, you can gain access to these additional details:

```
proc indent {} { return [string repeat { } $::indent] }
proc walkFrame {frame} {
   puts "[indent][$frame configure]"
   incr ::indent
   foreach sub [$frame subFrames] {
      walkFrame $sub
   }
   foreach inst [$frame instances] {
      if {![catch {$inst child} child]} {
         puts "[indent]INST [$inst configure name]"
         walkArch $child
      }
   }
   incr ::indent -1
}
proc walkArch {arch} {
   puts "[indent]ARCH [$arch configure fullName]"
   incr ::indent
   walkFrame [$arch frame]
   incr ::indent -1
   puts "[ indent]-- end [$arch configure fullName]"
}
set lib [library open uart]
set indent 0
walkArch [$lib declaration uart_tb struct]
```

Refer to “library object commands” on page 123, “instance object commands” on page 122 and “architecture object commands” on page 106 and “frame object commands” on page 120.

```
ARCH UART.uart_tb(struct)
 class architectureFrame name struct
  INST U_0
  ARCH UART.tester(flow)
   class architectureFrame name flow
```

```
    class machineFrame name tester_top
     class machineFrame name {tester_top test_rcv}
     class machineFrame name {tester_top test_status}
     class machineFrame name {tester_top test_xmit}
   -- end UART.tester(flow)
   INST U_1
   ARCH UART.uart_top(struct)
    class architectureFrame name struct
     INST U_1
     ARCH UART.cpu_interface(intconx)
      class architectureFrame name intconx
       class embeddedFrame name data_out_mux
       INST U_0
       ARCH UART.control_operation(fsm)
        class architectureFrame name fsm
         class machineFrame name cntrl_op
          class machineFrame name {cntrl_op RX}
          class machineFrame name {cntrl_op TX}
       -- end UART.control_operation(fsm)
     -- end UART.cpu_interface(intconx)
     INST U_2
     ARCH UART.clock_divider(flow)
      class architectureFrame name flow
       class machineFrame name clk_div
     -- end UART.clock_divider(flow)
     INST U_3
     ARCH UART.address_decode(tbl)
      class architectureFrame name tbl
     -- end UART.address_decode(tbl)
     INST U_4
     ARCH UART.serial_interface(struct)
      class architectureFrame name struct
       class embeddedFrame name convert
        class machineFrame name conv
       INST U_0
       ARCH UART.xmit_rcv_control(fsm)
        class architectureFrame name fsm
         class machineFrame name rcv
         class machineFrame name xmit
       -- end UART.xmit_rcv_control(fsm)
       INST U_1
       ARCH UART.status_registers(spec)
        class architectureFrame name spec
       -- end UART.status_registers(spec)
     -- end UART.serial_interface(struct)
   -- end UART.uart_top(struct)
-- end UART.uart_tb(struct)
```

Design Hierarchy and VHDL Configurations

The previous examples walked the design hierarchy assuming that each component instance was bound to a single architecture. In a design using VHDL configuration declarations, an architecture can have multiple configurations, each with different bindings for the component instances. Walking the hierarchy of such a design requires keeping track of the configuration.

The API includes configuration objects, which correspond to VHDL configuration declarations and frameConfiguration objects which correspond to a single level within the hierarchical structure of a configuration declaration. Designs containing FOR GENERATE, IF GENERATE and BLOCK statements will need corresponding entries in the configuration declarations, so when walking a design using configuration declarations, frames should be taken into account.

The following example shows how to walk a hierarchy using configuration declarations. It uses configuredSubFrames in place of subFrames and configuredChild in place of child.

```
proc indent {} { return [string repeat { } $::indent] }
proc walkFrame {frame frameCfg} {
   puts "[indent][$frame configure]"
   incr ::indent
   foreach {childFrame childFrameCfg} [$frame configuredSubFrames
$frameCfg] {
      walkFrame $childFrame $childFrameCfg
   }
   foreach inst [$frame instances] {
      if {![catch {$inst configuredChild $frameCfg} childInfo]} {
         foreach {childCfg childArch childFrameCfg} $childInfo {break}
         puts "[indent]INST [$inst configure name]"
         walkArch $childArch $childFrameCfg
      }
   }
   incr ::indent -1
}
proc walkArch {arch frameCfg} {
   puts "[indent]ARCH [$arch configure fullName]"
   incr ::indent
   walkFrame [$arch frame] $frameCfg
   incr ::indent -1
   puts "[ indent]-- end [$arch configure fullName]"
}
set lib [library open someLib]
set indent 0
set cfg [$lib declaration someCfg]
walkArch [$cfg architecture] [$cfg frameConfiguration]
```

Refer to “library object commands” on page 123, “instance object commands” on page 122 and “architecture object commands” on page 106 and “frame object commands” on page 120.

Issue: our sample designs do not use configurations.

Pruning the Hierarchy

The previous examples will expand each component instance, regardless of how many times it has previously been encountered within the hierarchy. In some cases, it is appropriate to prune the traversal so that a declaration is seen only once. This can be accomplished by adding a check at the start of walkArch:

```
proc walkArch {arch} {
   if {[info exists ::alreadyDone($arch)]} {
      return
```

```
    }

    puts "[indent]ARCH [$arch configure fullName]"
    incr ::indent
    walkFrame [$arch frame]
    incr ::indent -1
    puts "[ indent]-- end [$arch configure fullName]"
}
```

Finding the Compilation Order

Most VHDL compilation tools and some Verilog compilation tools need to be given the source files in an order which ensures that declarations are encountered before they are used. The API can be used to determine an appropriate order. The script shown below does this, though with some restrictions:

- For graphical VHDL designs, it assumes that the generation setting is for combined entity and architecture files.
- It does not handle VHDL configuration declarations, these require a more sophisticated form of pruning, since a single architecture can have a different structure for each configuration
- It does not handle HDL text files containing declarations that are not part of the design hierarchy

```
proc walkDependencies {decl} {
   global alreadyDone

   # Only look at each declaration once.
   if {[info exists alreadyDone($decl)]} {
      return
   }
   set alreadyDone($decl) 1

   # Only report each file once.
   set declFile [$decl file]
   if {[info exists alreadyDone($declFile)]} {
      set reportFile 0
   } else {
      set reportFile 1
      set alreadyDone($declFile) 1
   }

   foreach pkg [$decl packages] {
      walkDependencies $pkg
   }

   if {[$decl configure class] eq "architecture"} {
      walkDependencies [$decl entity]

      foreach inst [$decl instances] {
         if {![catch {$inst child} child]} {
            walkDependencies $child
         }
```

```
      }
   }

   set file [$decl file]
   set fileType [$file configure type]
   if {![regexp {Text$} $fileType]} {
      if {[lsearch {symbol blockInterface} $fileType] != -1} {
         # This assumes ent+arch are generated to a single file.
         set reportFile 0
      } else {
         set file [$file generated]
      }
   }

   if {$reportFile} {
      set lib [$file library]
      # Exclude standard and downstreamOnly libraries.
      if {[$lib configure type] eq "regular"} {
         puts "[$lib configure hardHdlDir]/[$file configure
relativePathname]"
      }
   }

   if {[$decl configure class] eq "packageHeader"} {
      walkDependencies [$decl body]
   }
}

catch {unset alreadyDone}
set lib [library open uart]
walkDependencies [$lib declaration uart_tb struct]
```

Refer to “library object commands” on page 123, “declaration object commands” on page 111, and “file object commands” on page 116.

```
/hds/install/UART/hdl/tester_flow.vhd
/hds/install/UART/hdl/control_operation_fsm.vhd
/hds/install/UART/hdl/cpu_interface_intconx.vhd
/hds/install/UART/hdl/clock_divider_flow.vhd
/hds/install/UART/hdl/address_decode_tbl.vhd
/hds/install/UART/hdl/xmit_rcv_control_fsm.vhd
/hds/install/UART/hdl/status_registers_spec.vhd
/hds/install/UART/hdl/status_registers_spec.vhd
/hds/install/UART/hdl/serial_interface_struct.vhd
/hds/install/UART/hdl/uart_top_struct.vhd
/hds/install/UART/hdl/uart_tb_struct.vhd
```

Chapter 2
HDS Tcl API Command Reference

Mentor Graphics has added a number of command extensions to the Tcl language to handle and support the HDL Designer operations. These commands are “built-in” and are executed the same as the standard Tcl commands.

This chapter provides a set of commands that can be used to work with and reference HDS objects. HDS Tcl commands are either pre-defined or dynamically created.

Pre-Defined Commands

General Commands

- addLibraryMapping (library args)
- configureProperties (designObject args)
- getDesignUnitFiles
- getLibrarriesRootDirectory
- getLibraryMacroDefinitions
- getLibrarySearchPath
- getLibraryVerilogIncludeSearchPath
- getVerilogIncludeSearchPath
- invokeGUI ()
- setBlackBoxFile
- setDefaultView(library file primary secondary)
- setDontTouchFile
- setLibrariesRootDirectory
- setLibraryMacroDefinitions
- setLibrarySearchPath
- setLibraryVerilogIncludeSearchPath
- setPolicyLocation (location)

- setRulesetLocation (location)
- setVerilogIncludeSearchPath
- unsetBlackBoxFile
- unsetDontTouchFile

addLibraryMapping

Usage:

addLibraryMapping <library> [-hdl <path>] [-hds <path>] [-downstream <path>] [-regular | -protected] [-all | -specified]

Description:

Adds (or changes) a library mapping and sets the library type and mode. Project file is edited and saved.

Arguments:

- library:

  Library name to add or change mapping(s) for.

- -hdl <path>:

  Adds (or changes) the HDL mapping and make it points to <path> location.

- -hds <path>:

  Adds (or changes) the HDS mapping and make it points to <path> location.

- -downstream <path>:

  Adds (or changes) a downstream mapping <mapping name> (example: ModelSim, QuestaSim, DesignChecker, etc.) and make it points to <path> location.

- -regular:

  Sets the library type to Regular. This option is the default.

- -protected:

  Sets the library type to Protected.

- -all:

  Sets the library mode to All. This option is the default.

- -specified:

  Sets the library mode to Specified.

Example:

Adding a new HDL & HDS mappings for a library named "mylib"

```
addLibraryMapping "mylib" -hdl "d:/mydesign" -hds "d:/libraries/mylib/hds"
-all
```

Adding a downstream mapping for "ModelSim Compile" task

```
addLibraryMapping "mylib" -downstream ModelSim "d:/libraries/mylib/work"
```

Changing the library type to Protected

```
addLibraryMapping "mylib" -protected
```

configureProperties

Set and get properties on a design object.

Syntax

configureProperties (designObject args)

Arguments

designObject	Design object: {library|library/file|library.unit ?primaryName? ?secondaryName? ?filePath?}.
args	Optional sequence of property keys and values: key ?value? ?key value? ....

Returns

Pass only a design object to return it's property names/values as a list {{name1} val1 {name2} val2}:

```
set properties [::configureProperties $designObject]
```

Pass a design object and a property name to get the value of the property on this object:

```
set propertyValue [::configureProperties $designObject
$propertyName]
```

Pass a design object and a list of property name/value pairs to set the value of properties on this object:

```
::configureProperties $designObject {$propertyName $propertyValue}
```

getDesignUnitFiles

Usage:

getDesignUnitFiles <unit> [-source | -text | -graphics] [-generated] [-defaultView] [-skipEntity] [-diskPath | -libraryPath]

Description:

Gets the files associated with a design unit.

Arguments:

- unit:

  The design unit name in the form "<library>.<unit>(<view>)" to get the associated files for specific view, or in the form "<library>.<unit>" to get the associated files for all/default view(s) (check "-defaultView:" option).

- -source:

  Default option. Get all source HDL and graphics files.

- -text:

  Get all source HDL files.

- -graphics:

  Get all source graphics files.

- -generated:

  Get all generated HDL files.

- -defaultView:

  Applicable only when no view name specified in the design unit name. Get the files associated with the unit's default view only. If not specified, get the files associated with all available view for the design unit.

- -skipEntity:

  Skip all files associated with the specified view's entity or symbol.

- -diskPath:

  Default option. Get the absolute file paths.

- -libraryPath:

  Get file paths relative to library HDL/HDS mappings.

Example:

Getting all source HDL and graphics files for "accumulator" design unit

```
getDesignUnitFiles "sequencer_vhd.accumulator" -source
```

Getting all graphical views files (except the symbol file) for "accumulator" design unit

```
getDesignUnitFiles "sequencer_vhd.accumulator" -graphics -skipEntity
```

Getting all generated files for "flow" view of "accumulator" design unit

```
getDesignUnitFiles "sequencer_vhd.accumulator(flow)" -generated
```

Getting absolute paths for all source files for "accumulator" design unit's default view

```
getDesignUnitFiles "sequencer_vhd.accumulator" -defaultView -absolute
```

Black-box all source HDL and graphics files for "accumulator" design unit

```
eval setBlackBoxFile "sequencer_vhd" [getDesignUnitFiles
"sequencer_vhd.accumulator" -source] -justification {"black box for DC"}
```

Returns:

List of design unit's associated files.

getLibrarriesRootDirectory

Usage:

getLibrariesRootDirectory

Description:

Gets the default root directory location for libraries' HDL and HDS mappings. The default root directory is used when there is no specific location specified for HDL or HDS mappings. Mappings directories are created under "<libraries root directory>/<library name>".

Example:

Printing the default libraries root directory

```
puts "Libraries default root directory: [getLibrariesRootDirectory]"
```

See Also:

- setLibrariesRootDirectory

getLibraryMacroDefinitions

Usage:

getLibraryMacroDefinitions <library> [-exact]

Description:

Gets the library macro definitions.

Arguments:

- <library>:

  Library name to get the macro definitions for.

- -exact:

  Return the exact macro definitions strings.

Example:

Printing the list of macro definitions for "mylib" library

```
set libMacros [getLibraryMacroDefinitions "mylib"]
foreach macro $libMacros {
     puts "Macro: $macro"
}
```

Getting the current exact macros string

```
set libMacros [getLibraryMacroDefinitions "mylib" -exact]
```

Append new macro definition to the currently defines ones

```
setLibraryMacroDefinitions "mylib" "$libMacros +define+WIDTH=8"
```

See Also:

- setLibraryMacroDefinitions

getLibrarySearchPath

Usage:

getLibrarySearchPath

Description:

Get the project libraries search path.

Example:

Printing the list of libraries in the libraries search path

```
set libSearchPath [getLibrarySearchPath]
puts "Libraries in library search path..."
foreach lib $libSearchPath {
     puts "Library: $lib"
}
```

See Also:

- setLibrarySearchPath

getLibraryVerilogIncludeSearchPath

Usage:

getLibraryVerilogIncludeSearchPath <library>

Description:

Gets the library Verilog include search path.

Arguments:

- <library>:

  Library name to get the search path for.

Example:

Printing the list of directories specified in the Verilog include search path for "mylib" library set library "mylib"

```
set libSearchPath [getLibraryVerilogIncludeSearchPath $library]
puts "Directories in library '$library' include search path..."
foreach dir $libSearchPath {
     puts "Include directory: $dir"
}
```

See Also:

- getVerilogIncludeSearchPath
- setLibraryVerilogIncludeSearchPath

getVerilogIncludeSearchPath

Usage:

getVerilogIncludeSearchPath

Description:

Gets the default Verilog include search path.

Example:

Printing the list of directories specified in the default Verilog include search path

```
set libSearchPath [getVerilogIncludeSearchPath]
puts "Directories in default include search path..."
foreach dir $libSearchPath {
     puts "Include directory: $dir"
}
```

See Also:

- setVerilogIncludeSearchPath
- getLibraryVerilogIncludeSearchPath

invokeGUI ()

Invoke the graphical user interface.

setBlackBoxFile

Usage:

setBlachBoxFile <library> <file 1> <file 2> <file 3>… [-diskPath | -libraryPath] [-justification <comment>]

Description:

Marks file(s) as black-box for DesignChecker task.

Arguments:

- library:

  Library name of the file(s) to be marked as black-box.

- file:

  Path of the file to be marked as black-box (check -diskPath:, and -libraryPath: options). Multiple files can be provided (separated by a white-space).

- -diskPath:

  Default option. Accept file paths as absolute or relative path on disk.

- -libraryPath:

  Accept file paths as relative to the library HDL/HDS mappings.

- -justification <comment>:

  Optional. Specifies a comment which explains why the file is marked as black-box.

Example:

Black-box HDL file in "uart_txt" library providing the absolute path on disk

```
setBlackBoxFile "uart_txt"
"$env(HDS_PROJECT_DIR)/uart_txt/hdl/clock_divider.v" -diskPath
```

Black-box two HDL files in "ethernet" library providing relative file paths

```
setBlackBoxFile "ethernet" "bench/verilog/tb_eth_behcomp.v"
"bench/verilog/stimuli/eth_txethmac_stim.v" -libraryPath
```

Black-box graphics file in "sequencer_vhd" library providing relative file path

```
setBlackBoxFile "sequencer_vhd" "accumulator/flow.fc" -libraryPath -
justification "black-box 'flow' flow chart view"
```

Black-box all source HDL and graphics files for "accumulator" design unit in "sequencer_vhd" library

```
eval setBlackBoxFile "sequencer_vhd" [getDesignUnitFiles
"sequencer_vhd.accumulator" -source] -justification {"black box for DC"}
```

See also:

unsetBlackBoxFile

setDefaultView

Set Default View.

Syntax

setDefaultView (library file primary secondary)

Arguments

library	Library name.
file	File path, relative to the library HDS mapping.
primary	Primary HDL declaration (VHDL entity or verilog module).
secondary	Secondary HDL declaration (VHDL architecture, use module name for verilog).

setDontTouchFile

Usage:

setDontTouchFile <library> <file 1> <file 2> <file 3>… [-diskPath | -libraryPath] [-tool <tool 1> <tool 2> <tool 3>…] [-justification <comment>]

Description:

Marks file(s) as dont touch for all tasks.

Arguments:

- library:

  Library name of the file(s) to be marked as dont touch.

- file:

  Path of the file to be marked as dont touch (check -diskPath:, and -libraryPath: options). Multiple files can be provided (separated by a white-space).

- -diskPath:

  Default option. Accept file paths as absolute or relative path on disk.

- -libraryPath:

  Accept file paths as relative to the library HDL/HDS mappings.

- -tool <tool 1> <tool 2> <tool 3>…:

  Specifies tools to mark this file as dont touch for. This argument is optional. Multiple tools can be specified. They could be one of the following:

ALL	All the tools. This is the default value.
H2G	Conversion to graphics
GENERATION	Generation
VHDL_CONFIGURATION_GENERATION	VHDL configuration generation
ALL_PLUGINS	All the downstream plug-ins (any plug-in which outputs files to downstream mapping).
SIMULATION_PLUGINS	All simulation plug-ins
SYNTHESIS_PLUGINS	All synthesis plug-ins
DESIGNCHECKER	DesignChecker plug-in
PLUGIN:<downstream plug-in name>	A specific downstream plug-in. Example" "PLUGIN:QuestaSim Compiler".

- -justification <comment>:

  Optional. Specifies a comment which explains why the file is marked as dont touch.

Example:

Dont touch HDL file in "uart_txt" library providing the absolute path on disk

```
setDontTouchFile "uart_txt"
"$env(HDS_PROJECT_DIR)/uart_txt/hdl/clock_divider.v" -diskPath
```

Dont touch two HDL files in "ethernet" library providing relative file paths

```
setDontTouchFile "ethernet" "bench/verilog/tb_eth_behcomp.v"
"bench/verilog/stimuli/eth_txethmac_stim.v" -libraryPath
```

Dont touch a HDL file in "ethernet" library for DesignChecker and QuestaSim Compiler plug-in only

```
setDontTouchFile "ethernet" "bench/verilog/tb_eth_behcomp.v" -libraryPath
-tool DESIGNCHECKER PLUGIN:QuestaSim\ Compiler
```

Dont touch graphics file in "sequencer_vhd" library providing relative file path

```
setDontTouchFile "sequencer_vhd" "accumulator/flow.fc" -libraryPath -
justification "dont touch 'flow' flow chart view"
```

Dont touch all source HDL and graphics files for "accumulator" design unit in "sequencer_vhd" library

```
eval setDontTouchFile "sequencer_vhd" [getDesignUnitFiles
"sequencer_vhd.accumulator" -source] -justification {"dont touch"}
```

See also:

unsetDontTouchFile

setLibrariesRootDirectory

Usage:

setLibrariesRootDirectory <directory>

Description:

Sets the default root directory location for libraries' HDL and HDS mappings. The default root directory is used when there is no specific location specified for HDL or HDS mappings. Mappings directories are created under "<libraries root directory>/<library name>".

Arguments:

- <directory>:

  Directory path where libraries mapping created underneath by default.

Example:

Setting the libraries default root directory preference

```
setLibrariesRootDirectory "d:/design/libraries"
```

Importing some files to a library that does not exist

```
runHdlImport "newlib" "d:/design/file1.v" "d:/design/file2.v"
```

Mappings (HDL and HDS) for library "newlib" are created at "d:/design/libraries/newlib"

See Also:

- getLibrarriesRootDirectory

setLibraryMacroDefinitions

Usage:

setLibraryMacroDefinitions <library> <macros>

Description:

Sets the library macro definitions.

Arguments:

- <library>:

  Library name to set the macro definitions for.

- <macros>:

  Macro definitions (separated by white-space or new line). Example: "+define+WIDTH=16 +define+ADDRESS=1".

Example:

Setting the macro definitions for "mylib" library

```
setLibraryMacroDefinitions "mylib" "+define+WIDTH=16 +define+ADDRESS=1"
```

See Also:

- getLibraryMacroDefinitions

setLibrarySearchPath

Usage:

setLibrarySearchPath <libraries> [-start | -end]

Description:

Sets the project libraries search path used when a missing component is referenced.

Arguments:

- <libraries>:

  List of libraries to be searched for missing components (separated by white-space). It can be empty. Example: "components_lib primitives_library".

- -start:

  Appends the provided search path to the beginning of the current search path.

- -end:

  Appends the provided search path to the end of the current search path.

Example:

Printing current library search path

```
puts "Current library search path \"[getLibrarySearchPath]\"."
```

Appending new libraries to the current search path

```
puts "Add 'components_lib' and 'primitives_lib' libraries to the search
path..."
setLibrarySearchPath "components_lib primitives_lib" -start
```

Printing the new library search path

```
puts "New library search path \"[getLibrarySearchPath]\"."
```

See Also:

- getLibrarySearchPath

setLibraryVerilogIncludeSearchPath

Usage:

setLibraryVerilogIncludeSearchPath <library> <directories> [-start | -end]

Description:

Sets the library Verilog include search path. The include search path is used to search for included files. The default include search path is used if this library-specific search path is not specified.

Arguments:

- <library>:

  Library name to set the search path for.

- <directory>:

  Search path directories (separated by ";"). Referencing environment variables and library names (where $<library name> points to the library's HDL mapping) is allowed. Example: "c:\my_includes;c:\design\includes".

- -start:

  Appends the provided search path to the beginning of the current search path.

- -end:

  Appends the provided search path to the end of the current search path.

Example:

Setting Verilog include search path for "mylib" library to path under the library's HDL mapping

```
setLibraryVerilogIncludeSearchPath "mylib" "\$mylib/includes"
```

Appending new directories to the Verilog include search path for "mylib" library

```
setLibraryVerilogIncludeSearchPath "mylib"
"c:\my_includes;c:\design\includes" -end
```

See Also:

- setVerilogIncludeSearchPath
- getLibraryVerilogIncludeSearchPath

setPolicyLocation

Set the location for DesignChecker policies.

Syntax

setPolicyLocation (location)

Arguments

location	Directory path beneath which the policies are located.

setRulesetLocation

Set the location for DesignChecker Rulesets.

Syntax

setRulesetLocation (location)

Arguments

location	Directory path beneath which the Rulesets are located.

setVerilogIncludeSearchPath

Usage:

setVerilogIncludeSearchPath <directories> [-start | -end]

Description:

Sets the default Verilog include search path. The include search path is used to search for included files. This path is used if the library specific include search path is not specified.

Arguments:

- <directory>:

  Search path directories (separated by ";"). Referencing environment variables and library names (where $<library name> points to the library's HDL mapping) is allowed. Example: "c:\my_includes;c:\design\includes".

- -start:

  Appends the provided search path to the beginning of the current search path.

- -end:

  Appends the provided search path to the end of the current search path.

Example:

Setting the default include search path preference

```
setVerilogIncludeSearchPath "c:\my_includes;c:\design\includes"
```

Append a new directory (specified by referencing an environment variable) to the default include search path preference

```
setVerilogIncludeSearchPath "\$MY_INCLUDES" -start
```

See Also:

- getVerilogIncludeSearchPath
- setLibraryVerilogIncludeSearchPath

unsetBlackBoxFile

Usage:

unsetBlackBoxFile <library> <file 1> <file 2> <file 3>… [-diskPath | -libraryPath]

Description:

Unsets the black-box property for file(s).

Arguments:

- library:

  Library name of the file(s) to have the black-box property unset for.

- file:

  Path of the file to have the black-box property unset for (check -diskPath:, and -libraryPath: options).

- -diskPath:

  Default option. Accept file paths as absolute or relative path on disk.

- -libraryPath:

  Accept file paths as relative to the library HDL/HDS mappings.

Example:

Remove black-box property for HDL file in "uart_txt" library providing the absolute path on disk

```
unsetBlackBoxFile "uart_txt"
"$env(HDS_PROJECT_DIR)/uart_txt/hdl/clock_divider.v" -diskPath
```

Remove black-box property for two HDL files in "ethernet" library providing relative file paths

```
unsetBlackBoxFile "ethernet" "bench/verilog/tb_eth_behcomp.v"
"bench/verilog/stimuli/eth_txethmac_stim.v" -libraryPath
```

Remove black-box property for graphics file in "sequencer_vhd" library providing relative file path

```
unsetBlackBoxFile "sequencer_vhd" "accumulator/flow.fc" -libraryPath
```

See also:

setBlackBoxFile

unsetDontTouchFile

Usage:

unsetDontTouchFile <library> <file 1> <file 2> <file 3>… [-diskPath | -libraryPath] [-tool <tool 1> <tool 2> <tool 3>…]

Description:

Unsets the dont touch property for file(s).

Arguments:

- library:

  Library name of the file(s) to have the dont touch property unset for.

- file:

  Path of the file to have the dont touch property unset for (check -diskPath:, and -libraryPath: options). Multiple files can be provided (separated by a white-space).

- -diskPath:

  Default option. Accept file paths as absolute or relative path on disk.

- -libraryPath:

  Accept file paths as relative to the library HDL/HDS mappings.

- -tool <tool 1> <tool 2> <tool 3>…:

  Specifies tools to unmark this file as dont touch for. This argument is optional. Multiple tools can be specified. They could be one of the following:

ALL	All the tools. This is the default value.
H2G	Conversion to graphics
GENERATION	Generation
VHDL_CONFIGURATION_GENERATION	VHDL configuration generation
ALL_PLUGINS	All the downstream plug-ins (any plug-in which outputs files to downstream mapping).
SIMULATION_PLUGINS	All simulation plug-ins
SYNTHESIS_PLUGINS	All synthesis plug-ins
DESIGNCHECKER	DesignChecker plug-in
PLUGIN:<downstream plug-in name>	A specific downstream plug-in. Example" "PLUGIN:QuestaSim Compiler".

Example:

Remove dont touch property for HDL file in "uart_txt" library providing the absolute path on disk

```
unsetDontTouchFile "uart_txt"
"$env(HDS_PROJECT_DIR)/uart_txt/hdl/clock_divider.v" -diskPath
```

Remove dont touch property for two HDL files in "ethernet" library

```
unsetDontTouchFile "ethernet" "bench/verilog/tb_eth_behcomp.v"
"bench/verilog/stimuli/eth_txethmac_stim.v" -libraryPath
```

Remove dont touch property for a HDL file in "ethernet" library for Generation and DesignChecker

```
unsetDontTouchFile "ethernet" "bench/verilog/tb_eth_behcomp.v" -
libraryPath -tool GENERATION DESIGNCHECKER
```

Remove dont touch property for graphics file in "sequencer_vhd" library

```
unsetDontTouchFile "sequencer_vhd" "accumulator/flow.fc" -libraryPath
```

See also:

setDontTouchFile

DesignChecker Commands

The following DesignChecker commands cannot be used directly in HDS batch files. These commands can only be used in a Tcl file passed to an HDS batch file through the ConfigFile option as follows:

```
setupTask {DesignChecker} –settings ConfigFile {<absolute path of the tcl
file >}
```

- disableDumpCodeSnippet()
- disableFilterAssocViolation()
- enableDumpCodeSnippet ()
- enableFilterAssocViolation ()
- setCheckedFileDUsReport (filelist formatlist)
- setCheckedFileDUsReportContents (panes)
- setExclusionReport (filelist formatlist)
- setExclusionReportContents (panes)
- setRulesCheckedReport (filelist formatlist)
- setSnippetLinesAfter (linesCount)
- setSnippetLinesBefore (linesCount)

disableDumpCodeSnippet()

Disable the dumping of code snippets along with violations in the results report produced by DesignChecker.

disableFilterAssocViolation()

Disable the filtering of associated violations while dumping the results report generated by DesignChecker.

enableDumpCodeSnippet ()

Allow code snippets to be dumped along with violations in the results report produced by DesignChecker.

enableFilterAssocViolation ()

Enable filtering associated violations while dumping the results report. This affects the results report generated by DesignChecker.

setCheckedFileDUsReport

Set the list of files, and the corresponding formats, in which the Checked File/Design Units reports will be dumped. This affects the Checked File/Design Units report generated by DesignChecker.

Syntax

setCheckedFileDUsReport (filelist formatlist)

Arguments

filelist	A list of the files to which Checked File/Design Units reports can be dumped. Using this API, you can generate multiple Checked File/Design Units reports.
formatlist	The corresponding format of each report file. For example: `setCheckedFileDUsReport {"/home/username/CheckedFiles.htm"} {"HTML"}`

setCheckedFileDUsReportContents

Set the contents of the checked file/design units report, that is, specify which panes of the DesignChecker Checked File/Design Units tab will be dumped in the report.

Syntax

setCheckedFileDUsReportContents (panes)

Arguments

panes	List of panes which need to be dumped in the Checked File/Design Units report. For example: `setCheckedFileDUsReportContents {{Checked Files} {Checked Design Units}}`

setExclusionReport

Set the list of files, and the corresponding formats, in which the exclusion reports will be dumped. This affects the exclusion reports generated by DesignChecker,

Syntax

setExclusionReport (filelist formatlist)

Arguments

filelist	A list of the files to which exclusion reports can be dumped. Using this API, you can generate multiple exclusion reports. For example: `setExclusionReport {"/home/username/ExcRep.csv" "/home/username/ExcRep.htm"} {"CSV" "HTML"}`
formatlist	The corresponding format of each report file.

setExclusionReportContents

Set the contents of the exclusion report, that is, specify which panes of the DesignChecker Exclusions tab will be dumped in the exclusions report.

Syntax

setExclusionReportContents (panes)

Arguments

panes	List of panes which need to be dumped in the exclusions report. For example: `setExclusionReportContents {{Code/Rule Exclusions} {Black Boxed Files} {Don't Touch Files} {Exclusion Pragmas} {Pragma Code Excluded} {Unbound Component/Instances} {Summary}}`

setRulesCheckedReport

Set the list of files, and the corresponding formats, in which the rule details reports will be dumped. This affects the rule details report generated by DesignChecker.

Syntax

setRulesCheckedReport (filelist formatlist)

Arguments

filelist	A list of the files to which rule details reports can be dumped. Using this API, you can generate multiple rule details reports. For example: `setRulesCheckedReport {"/home/username/RulesCheckedRep.csv" "/home/username/RulesCheckedRep.tsv"} {"CSV" "TSV"}`
formatlist	The corresponding format of each report file.

setSnippetLinesAfter

Specify the number of lines to add to the dumped code snippet after the violation line number. This affects the code snippet displayed in the results report generated by DesignChecker.

Syntax

setSnippetLinesAfter (linesCount)

Arguments

linesCount	The number of lines to add to the dumped code snippet after the violation line number.

setSnippetLinesBefore

Specify the number of lines to add to the dumped code snippet before the violation line number. This affects the code snippet displayed in the results report generated by DesignChecker.

Syntax

setSnippetLinesBefore (linesCount)

Arguments

linesCount	The number of lines to add to the dumped code snippet before the violation line number

HDL2Graphics Commands

- runH2G (hierarchyType library file primary secondary)
- runHdlImport (library args)
- runHtmlExport (library unit view file)
- setupH2G (args)
- setupHdlImport (args)
- setupHtmlExport (args)

runH2G

Convert specified text declarations to graphics. Uses options set by setupH2G.

Syntax

runH2G (hierarchyType library file primary secondary)

Arguments

hierarchy	Type SINGLE | HIERARCHY.
library	Library name.
file	File path, relative to the library HDL mapping.
primary	Primary HDL declaration (VHDL entity or verilog module).
secondary	Secondary HDL declaration (VHDL architecture).

runHdlImport

Usage:

runHdlImport <library> [<path to file 1> … <path to file n>] [-filelist <path to filelist>] [-copy | -point] [-clean]

Description:

Imports HDL files to a new or an existing library. Options set by setupHdlImport are used.

Arguments:

- <library>:

  The name of the target library where files are imported. It could be an empty string. If an empty string is provided (and no library specified in filelist), the project's default library is used if -copy option is supplied and a made-up library name is used if -point option is supplied. If the library does not exist, it is automatically created at the libraries root directory preference (see setLibrariesRootDirectory).

- <path to file 1> … <path to file n>:

  File paths to be imported. File paths can be specified relative to the current working directory.

- -filelist <path to file-list>:

  Imports a list of files specified in a filelist (see HDS User Manual for filelist format). Filelist path can be specified relative to the current working directory.

- -copy:

  Imports the files by copying them to the target library HDL mapping. This option is the default.

- -point:

  Imports the files by pointing the library HDL mapping to them. If the target library is already created, the HDL mapping must by enclosing all the files to import.

- -clean:

  Applicable only when -point option is supplied. If the target library is not created, it cleans the HDS mapping specified by the libraries root directory preference (see setLibrariesRootDirectory) if it is not empty.

Example:

Importing files by pointing to them (without copying)

```
runHdlImport "mylib" "d:/mydesign/file1.v" "d:/mydesign/file2.v" -point -clean
```

Importing files specified in a file-list by copying them to the target library location

```
runHdlImport "mylib" -filelist "d:/mydesign/components/filelist.txt" -
copy
```

Importing a set of files specified in a Tcl list

```
set files {{d:/mydesign/file1.v} {d:/mydesign/file2.v}}
eval runHdlImport {""} $files -point -clean
```

See Also:

- setupHdlImport

runHtmlExport

Run the HTML export on the given design object. Uses options set by setupHtmlExport.

Syntax

runHtmlExport (library unit view file)

Arguments

library	Library name.
unit	Design unit name.
view	View name for graphics or secondary declaration name for HDL (VHDL architecture). View name can be left empty when exporting primary unit files (e.g. graphical symbol).
file	File path, relative to the library HDS/HDL mapping.

setupH2G

Setup the options for runH2G.

Syntax

setupH2G (args)

Arguments

args	Option names and values.
View Styles Options:	
createAltSmFlowChart ON|OFF (ON)	If a State Diagram view cannot be created, create a FlowChart view instead.
createBlockDiagram ON|OFF (ON)	Create a structural Block Diagram view.
createFlowChart ON|OFF (OFF)	Create a Flow Chart view.
createIBD ON|OFF (OFF)	Create an IBD view.
createLeafBlockDiagram ON|OFF (OFF)	Create a leaf level Block Diagram view.
createStateDiagram ON|OFF (OFF)	Create a State Diagram view.
createSymbolAlways ON|OFF (OFF)	Create a symbol even if no view styles are selected.
General Options:	
overwrite ON|OFF (OFF)	If a graphical view already exists, overwrite it.
copyOnOverwrite ON|OFF (OFF)	Copy a view before overwriting it.
searchLibraries <libraries>	Search for black box components in these libraries.
setDefaultView ON|OFF (ON)	Set graphical views as default.
setLibrary <unit> <library>	Specify a library for a black box component.
verbose ON|OFF (OFF)	Display additional information during conversion.
setCreateForGenerate ON|OFF (ON)	Create graphical views rather than visualizations.
Block Diagram Options:	
assignmentShape RECTANGLE|BUFFER (RECTANGLE)	Specify the shape of concurrent assignments, if extracted.

connectEmbedded ON|OFF (OFF)	Create signals between embedded blocks.
createEmbeddedStateDiagram ON|OFF (OFF)	Create embedded state machines if possible.
defaultInstanceView ON|OFF (OFF)	Apply to instances in preference to a specified view.
diagramTextVisibility NONE|ALL|PREFS (NONE)	Specify the visibility of text associated with the diagram.
extractAssignments ON|OFF (OFF)	Create separate embedded blocks for concurrent assignments.
instanceLimit <positive integer> (1000)	Maximum number of instances allowed in a Block Diagram.
netTextVisibility NONE|PORT_ONLY|ALL|PREFS (PORT_ONLY)	Specify the visibility of text associated with nets.
portTextVisibility NONE|NAME_ONLY|NAME_AND_TYPE|PREFS (PREFS)	Specify the visibility of text associated with PortIO.
Placement Options:	
instanceAutoSize WIDTH|HEIGHT|WIDTH_AND_HEIGHT|NONE (WIDTH_AND_HEIGHT)	Size component instances to fit the visible text.
placementLimit <positive integer> (50)	Maximum number of auto-placed instances.
preservePlacement ON|OFF (OFF)	Preserve placement of existing diagrams.
Routing Options:	
alignPorts ON|OFF (ON)	Align port I/O with connections.
bundleLimit <positive integer> (5)	Maximum number of signals allowed in a bundle.
bundleSignals DIRECT|INDIRECT|NONE (NONE)	Bundle signals between nodes.
busReconstruct ON|OFF (OFF)	Reconstruct buses for signal slices.
connectionLimit <positive integer>	Maximum number of connections allowed for a net.

globalConnectors ON|OFF (ON)	Create global connectors for global signals.
movePorts ON|OFF (ON)	Allow connector end points to move during routing.
routeSignals ON|OFF (ON)	Connect and route signals on the diagram.
useDefaults	Set all options to default values. Default values are shown in brackets, above.

setupHdlImport

Usage:

setupHdlImport [-useDefaults] [-overwrite (ON | OFF)] [-importReferencedText (ON | OFF)] [-importDirectoryStructure (ON | OFF)] [-defaultLanguage (VHDL | Verilog | UNKNOWN)]

Description:

Setup the options for runHdlImport.

Arguments:

- -useDefaults:

  Sets all options to default values. Default values are shown in brackets, above.

- -overwrite:

  Overwrite existing files.

- -importReferencedText:

  Reference files from their original location instead of copying.

- -importDirectoryStructure:

  Imports directory structure of source files.

- -defaultLanguage:

  Specifies the language of source files with unrecognized extensions. Use UNKNOWN to import non-HDL files.

setupHtmlExport

Setup the options for runHTMLExport.

Syntax

setupHtmlExport (args)

Arguments

args	Option names and values.
HTML Settings:	
-reset_defaults	Reset all options back to their defaults.
-export_directory <path>	Target directory for export.
-hierarchy_levels ALL|<number> (ALL)	Levels of hierarchy to descend.
-export_visualizations ON|OFF (ON)	Switch to creation of visualization.
-include_doc_types ALL|<doc_types> (ALL)	Add a list of document types to include in the export (see the Documentation and Visualization Options dialog box for document type names).
-include_only_doc_types	Make these document types the only ones included for export. More types can be added with subsequent include_doc_types calls (see the Documentation and Visualization Options dialog box for document type names).
-export_all_ict ON|OFF	Switch to export all ICT views.
-export_generated ON|OFF (ON)	Switch to export generated files.
-export_sidedata_files ALL|REGISTERED (ALL)	Set the types of sidedata files to export.
-export_sidedata_type USER|DESIGN|ALL|NONE (ALL)	Set the types of sidedata to export.
-title_page AUTO|<path> (AUTO)	Set the path to the title page to use.
-index_page AUTO|<page> (AUTO)	Set the index page name.
Graphics Settings:	
-graphics_format JPEG|PNG|SVG (SVG)	Set the format to use when exporting graphics.
-jpeg_quality <1- 100> (100)	Quality to use for JPEG graphics.

-graphics_size <1- 100> (100)	Percentage size for graphics.

Batch Mode Tasks

- runTask(args)
- setCompileAlways (enable)
- setupTask (args)

Use these commands to setup and run tasks in batch mode.

When specifying a task, be aware that it is a tcl list, with each element of the list being a level of hierarchy

As each element is separated with whitespace, be sure to escape whitespace that is part of a task name

For example, if you have a flow arranged like this:

Flow 1

|--- Tool A

|--- Tool B

|--- Flow 2

    |--- Tool C

    |--- Tool D

This refers to "Flow 1" : {Flow\ 1}

This refers to "Tool A" : {Flow\ 1 Tool\ A}

This refers to "Tool D" : {Flow\ 1 Flow\ 2 Tool\ D}

runTask

Run a task.

Syntax

runTask (args)

Arguments

args	Option names and values.
<taskName>	Specify the task run. Must be specified.
[library [unit [view [pathName]]]]	Specifies the view to run the task on. If none are specified uses design root.
user	If HDS finds a user and a team task with the same name, pick the user task. This is the default.
team	If HDS finds a user and a team task with the same name, pick the team task.

setCompileAlways (enable)

Enable or disable incremental compilation for all incremental compilers.

Syntax

setCompileAlways (enable)

Arguments

enable	ON|OFF: Set the global compile always switch (defaults to off)

setupTask (args)

Setup options for a task.

Syntax

setupTask (args)

Arguments

<taskName>	Specify the task to apply these setup options to. Must be specified.
setting <name> <value>	Changes the value of the setting given by <name> on the specified task.
single	Changes the hierarchy depth to be used when running the task to single level.
hierarchical	Changes the hierarchy depth to be used when running the task to hierarchy through blocks.
throughCpt	Changes the hierarchy depth to be used when running the task to hierarchy through components.
user	If HDS finds a user and a team task with the same name, pick the user task. This is the default.
team	If HDS finds a user and a team task with the same name, pick the team task.

Generation Commands

- runConfigGenerate
- runGenerate
- setupGenerate

runConfigGenerate

Generate VHDL configuration files for the specified graphical objects. Uses options set by setupGenerate.

Syntax

runConfigGenerate (library entity arch)

Arguments

library (= {})	Library name.
entity (= {})	Design unit name for graphics.
arch (= {})	View name (file name without extension) for graphics file.

runGenerate

Generate HDL for the specified graphical objects. Uses options set by setupGenerate.

Syntax

runGenerate (library entity arch)

Arguments

library (= {})	Library name. (If you want to generate a whole library, then specify this value only).
entity (= {})	Design unit name for graphics.
arch (= {})	View name (file name with extension) for graphics file.

setupGenerate

Setup options for runGenerate and runConfigGenerate.

Syntax

setupGenerate (args)

Arguments

useDefaults	Using this value will allow you to override your preference values to the following.
generateAlways ON|OFF	Always generate even if no changes have been made. Default (OFF).
semanticChecking ON|OFF	Default (ON).
semanticWarnings ON|OFF	Default (OFF).
tabWidth (POSITIVE INTEGER)	Default (3).
autoTypeConversion NONE | CONV_FUNC_ONLY | FULL>	Default (FULL).
combineEntityArch ON|OFF	Default (OFF).
embedConfigStatements ON|OFF	Default (ON).
embedViewNameInConfig ON|OFF	Default (ON).
embedViewNameInConfig ON|OFF	Default (OFF).
hierarchicalConfigSpecification ON|OFF	Default (OFF).
thruCptsConfigSpecification ON|OFF	Default (OFF).
synopsysPragmasForEmbeddedConfig ON|OFF	Default (ON).
single	This is the default.
hierarchical	Goes down through blocks.
throughCpt	Goes down through blocks and components.
genericsInConfig ON|OFF	Default (OFF).
workAsLibInConfig ON|OFF	Default (OFF).
leafInConfig ON|OFF	Default (OFF).
viewInStandaloneConfig ON|OFF	Default (OFF).

customCodeGen ON|OFF	Default (OFF). If you require a flat configuration file, when using runConfigGenerate specify -hierarchicalConfigSpecification and -thruCptsConfigSpecification with OFF.
useDefaultCustomCodeScript ON|OFF	Default (ON). Specify whether you want to use the default custom code script.
includeVhdlGenProps ON|OFF	Include generation properties after header in generated VHDL.
includeVerilogGenProps ON|OFF	Include generation properties after header in generated Verilog.

Library Commands

- library names()
- library open

library names()

Obtains the names of all libraries in the project.

Syntax

library names()

Arguments

entity (= {})	Design unit name for graphics.
arch (= {})	View name (file name without extension) for graphics file.

Returns

List of library names.

library open

Opens a library so its content can be examined.

Syntax

library open (libraryName)

Arguments

libraryName	The library to open.

Returns

The library object, see “library object commands” on page 123.

Version Management Commands

- runVMChangeLock (args)
- runVMCheckIn (args)
- runVMCheckOut (args)
- runVMGet (args)
- runVMHistory (args)
- runVMLabel (args)
- runVMSynchronize (args)
- runVMUndoCheckOut (args)
- setupVM (option arg)
- setupVMChangeLock (option arg)
- setupVMCheckIn (option arg)
- setupVMCheckOut (option arg)
- setupVMGet (option arg)
- setupVMHierarchy (option arg)
- setupVMLabel (option arg)
- setupVMSynchronize (option arg)

runVMChangeLock

Change lock on a specified library, design unit, view. Uses options set by setupVMChangeLock.

Syntax

runVMChangeLock (args)

Arguments

library	The library name. This is required.
view	View name (optional, absent if design unit not specified).

runVMCheckIn

Check in a specified library, design unit, view. Uses options set by setupVMCheckIn.

Syntax

runVMCheckIn (args)

Arguments

library	The library name. This is required.
designUnit	Design unit name (optional).
view	View name (optional, absent if design unit not specified).

runVMCheckOut

Check out a specified library, design unit, view. Uses options set by setupVMCheckOut.

Syntax

runVMCheckOut (args)

Arguments

library	The library name. This is required.
designUnit	Design unit name (optional).
view	View name (optional, absent if design unit not specified).

runVMGet

Get a specified library, design unit, view. Uses options set by setupVMGet.

Syntax

runVMGet (args)

Arguments

library	The library name. This is required.
designUnit	Design unit name (optional).
view	View name (optional, absent if design unit not specified).

runVMHistory

Get the history of a specified library, design unit, view.

Syntax

runVMHistory (args)

Arguments

library	The library name. This is required.
designUnit	Design unit name (optional).
view	View name (optional, absent if design unit not specified).

runVMLabel

Label a specified library, design unit, view. Uses options set by setupVMLabel.

Syntax

runVMLabel (args)

Arguments

library	The library name. This is required.
designUnit	Design unit name (optional).
view	View name (optional, absent if design unit not specified).

runVMSynchronize (args)

Synchronize a specified library, design unit, view. Uses options set by setupVMSynchronize.

Syntax

runVMSynchronize (args)

Arguments

library	The library name. This is required.
designUnit	Design unit name (optional).
view	View name (optional, absent if design unit not specified).

runVMUndoCheckOut

Undo the check out of a specified library, design unit, view.

Syntax

runVMUndoCheckOut (args)

Arguments

library	The library name. This is required.
designUnit	Design unit name (optional).
view	View name (optional, absent if design unit not specified).

setupVM

Setup the options for Version Management.

Syntax

setupVM (option arg)

Arguments

option	Option name.
arg	Option value.
tool <string>	The version management interface required.
library <string>	Specify the library on which you want to invoke these options (this is only necessary if you want to set the repository mappings).
repository <string>	Specify the repository mapping.
hds_repository <string>	Specify the hds repository mapping.
hdl_repository <string>	The hdl repository mapping.
include_default_view 1|0	You can choose to version manage the default view file.
include_side_data 1|0	Side data for a version managed object is included.
verbose 1|0	Additional info is displayed in the log window.
create_empty_side_data 1|0	Automatically create side data directories if they do not exist.
multiple_repository_mode 1|0	Repository is defined on a mapping by mapping basis.
include_generated_hdl 1|0	Generated HDL for a version managed object is included.

setupVMChangeLock

Setup the change lock options for Version Management.

Syntax

setupVMChangeLock (option arg)

Arguments

option	Option name.
arg	Option value.
version <string>	The version required.
change_lock 1|0	When you lock an object, the version in your private workspace is made editable. If you unlock an object, the version in your workspace is made read-only.

setupVMCheckIn

Setup the check-in options for Version Management.

Syntax

setupVMCheckIn (option arg)

Arguments

option	Option name.
arg	Option value.
version <string>	The version required.
label <string>	You can specify a symbolic label which is associated with a particular set of objects.
description <string>	Specify a description for the check in.
overwrite_label 1|0	Transfer the label to the new version number.
retain_lock 1|0	You can choose to retain a lock on the checked in objects.

setupVMCheckOut

Setup the check-out options for Version Management.

Syntax

setupVMCheckOut (option arg)

Arguments

option	Option name.
arg	Option value.
version <string>	The version required.
lock 1|0	The objects are normally checked out with a lock to prevent concurrent editing by other users but you can choose to unset this option.
comment <string>	If you are using ClearCase, you can enter a comment in the description field.

setupVMGet

Setup the Get options for Version Management.

Syntax

setupVMGet (option arg)

Arguments

option	Option name.
arg	Option value.
version <string>	The version required.
overwrite 1|0	Transfer the label to the new version number.

setupVMHierarchy

Setup the hierarchy options for Version Management.

Syntax

setupVMHierarchy (option arg)

Arguments

option	Option name.
arg	Option value.
hierarchy 1|0	Include the hierarchy below the selected object.
through_components 1|0	If hierarchy selected, then include components.
through_libraries 1|0	You can choose to limit the scope to the current library or operate on objects through all libraries.
include_protected_libraries 1|0	Include objects in protected libraries.
packages 1|0	Automatically include referenced VHDL packages or Verilog include files.

setupVMLabel

Setup the label options for Version Management.

Syntax

setupVMLabel (option arg)

Arguments

option	Option name.
arg	Option value.
version <string>	The version required.
label <string>	Specify the required label.
add 1|0	Add or remove the label.
overwrite 1|0	Transfer the label to the new version number.
description	Specify a description for the label.

setupVMSynchronize

Setup the Synchronize options for Version Management.

Syntax

setupVMSynchronize (option arg)

Arguments

option	Option name.
arg	Option value.
version <string>	The version required.
add_files 1|0	If this option is not set, only objects that already exist are synchronized.

Dynamically Created Commands

Command Structure

A dynamically created Tcl command consists of an HDS object or a variable holding the value of a handle to that object. The object is followed by an option possibly followed by one or more arguments and is written according to the following format:

HDS object /Option/?Arguments

HDS objects

HDS Objects listing

architecture - block - blockFrame - caseFrame - configuration - declaration - elseFrame - embeddedFrame - file - forFrame - frameConfiguration - frame - ifFrame - instance - library - machineFrame - package Body

Example

```
$lib configure class
```

The command in the above example is a library command in other words the HDS object referenced in this command is a library object. The variable **$lib** holds the value of a handle referring to the UART library.

Variables are created using the standard Tcl set command. HDS refers to objects through a set of handles.The pre-defined library open command returns a handle to the UART library that substitutes the value of the lib variable.

```
set lib [library open uart]
```

A variable can hold the value of one or more handles referring to HDS objects i.e

```
set fl [$lib files]
```

While the variable **$lib** holds the value of the handle hds0 which refers to the library UART, variable **$fl** holds the values of a set of handles referring to the UART library files.

Notice that variable names are always preceded by a $.

Option

Options Listing

configure - entity - file - frame instances - packages - configuredSubFrames - subFrames - architecture - configuredChild - close - declaration - files - hdlDirectory - packages

Example

$lib configure class

All objects have a ‘configure’ command option, this provides access to the attributes of the object. If you use the configure command with no arguments, it shows the names and values of all the attributes

Arguments

Each command may have one or more argument

Command list

architecture - block - blockFrame - caseFrame - configuration - declaration - elseFrame - embeddedFrame - file - forframe - frameConfiguration - frame - ifframe - instance - library - machineFrame - package Body

architecture object commands

Associated with an implementation of an entity. Corresponds to a VHDL architecture, a Verilog module and an HDS structural or leaf graphics file.

- $architecture configure (args)
- $architecture entity ()
- $architecture file ()
- $architecture frame ()
- $architecture instances ()
- $architecture packages ()

Table 2-1. architecture object commands

Command	Description
$architecture configure (args)	With no arguments lists the names and values of all options. The following arguments are available: **class** : identifies the object as an architecture **name** : name of the declaration itself **fullName** : fully qualified name including those of parent objects **startLine** : the line number where the declaration begins in the source file **refName** : Name that can be to refer to the object when running tasks or accessing properties
$architecture entity ()	Gets the entity associated with the architecture. Refer to “entity option object commands” on page 114. Returns: The entity object.
$architecture file ()	Obtains the file containing the object. Returns: A single object, refer to“file object commands” on page 116.
$architecture frame ()	Obtains the root frame of the architecture. Refer to “frame object commands” on page 120. Returns: The root frame.
$architecture instances ()	Gets all the instances in all frames of the architecture. Refer to “instance object commands” on page 122.
$architecture packages ()	Obtains the packages referenced. Returns: A list of packages, refer to “packageHeader object commands” on page 126, or an empty list if the declaration is not VHDL.

blockFrame object commands

A VHDL block statement

- $blockFrame configure (args)
- $blockFrame configuredSubFrames (frameConfiguration)
- $blockFrame instances ()
- $blockFrame subFrames ()

Table 2-2. blockFrame object commands

Command	Description
$blockFrame configure (args)	With no arguments lists the names and values of all options. With one argument lists the value of the specified option. **Arguments** **class** : identifies the object as a blockFrame **name** :
$blockFrame configuredSubFrames (frameConfiguration)	Obtains the sub-frames. **Arguments** **frameConfiguration** : defines a level in a VHDL configuration declaration from which the bindings are selected **Returns** A list of sub-frame information, each consecutive pair represents a frame and a frameConfiguration in that order.
$blockFrame instances ()	Obtains the component instances; Instances contained within a sub-frame are not included. Refer to “instance object commands” on page 122. **Returns** A list of instances.
$blockFrame subFrames ()	Obtains the sub-frames. **Returns** A list of sub-frames.

caseFrame object commands

A Verilog case statement

- $caseFrame configure (args)
- $caseFrame configuredSubFrames (frameConfiguration)
- $caseFrame instances ()
- $caseFrame subFrames ()

Table 2-3. caseFrame object commands

Command	Description
$caseFrame configure (args)	With no arguments lists the names and values of all options. With one argument lists the value of the specified option. **Arguments** **class** : identifies the object as a caseFrame **name** :
$caseFrame configuredSubFrames (frameConfiguration)	Obtains the sub-frames. **Arguments** **frameConfiguration** :defines a level in a VHDL configuration declaration from which the bindings are selected **Returns** A list of sub-frame information, each consecutive pair represents a frame and a frameConfiguration in that order.
$caseFrame instances ()	Obtains the component instances, directly contained within the frame. Instances contained within a sub-frame are not included. Refer to "instance object commands" on page 122. **Returns** A list of instances.
$caseFrame subFrames ()	Obtains the sub-frames. **Returns** A list of sub-frames.

configuration object commands

A VHDL configuration declaration

- $configuration architecture ()
- $configuration configure (args)
- $configuration file ()
- $configuration frameConfiguration ()
- $configuration packages ()

Table 2-4. configuration object commands

Command	Description
$configuration architecture ()	Gets the architecture associated with the configuration. Refer to"architecture object commands" on page 106. **Returns** The architecture object.
$configuration configure (args)	With no arguments lists the names and values of all options. With one argument lists the value of the specified option. **Arguments** **class** : identifies the object as a configuration **name** : name of the declaration itself **fullName** : fully qualified name including those of parent objects **startLine** : the line number where the declaration begins in the source file **refName** : Name that can be to refer to the object when running tasks or accessing properties
$configuration file ()	Obtain the file containing the object. **Returns** A single object. Refer to "file object commands" on page 116.

Table 2-4. configuration object commands (cont.)

Command	Description
$configuration frameConfiguration ()	Gets the frame configuration associated with the configuration, refer to "frameConfiguration object commands" on page 119. **Returns** The architecture object.
$configuration packages ()	Obtains the packages referenced. **Returns** A list of packages, refer to "packageHeader object commands" on page 126, or an empty list if the declaration is not VHDL.

declaration object commands

- $declaration configure (args)
- $declaration file ()
- $declaration packages ()

Table 2-5. declaration object commands

Command	Description
$declaration configure (args)	With no arguments lists the names and values of all options. With one argument lists the value of the specified option. **Arguments** **name** : name of the declaration itself **fullName** : fully qualified name including those of parent objects **startLine** : the line number where the declaration begins in the source file **refName** : Name that can be to refer to the object when running tasks or accessing properties **class** : identifies the object as a declaration
$declaration file ()	Obtains the file containing the object. **Returns** A single object, refer to “file object commands” on page 116.
$declaration packages ()	Obtains the packages referenced. **Returns** A list of packages, see “packageHeader object commands” on page 126, or an empty list if the declaration is not VHDL.

elseFrame object commands

A Verilog `else statement

- $elseFrame configure (args)
- $elseFrame configuredSubFrames (frameConfiguration)
- $elseFrame instances ()
- $elseFrame subFrames ()

Table 2-6. elseFrame object commands

Command	Description
$elseFrame configure (args)	With no arguments lists the names and values of all options. With one argument lists the value of the specified option. **Arguments** **class** : identifies the object as an elseFrame **name** :
$elseFrame configuredSubFrames (frameConfiguration)	Obtains the sub-frames. **Argument** **frameConfiguration** :defines a level in a VHDL configuration declaration from which the bindings are selected **Returns** A list of sub-frame information, each consecutive pair represents a frame and a frameConfiguration in that order.
$elseFrame instances ()	Obtains the component instances, refer to"instance object commands" on page 122, directly contained within the frame. Instances contained within a sub-frame are not included. **Returns** A list of instances.
$elseFrame subFrames ()	Obtains the sub-frames. **Returns** A list of sub-frames.

embeddedFrame object commands

Represents an embedded text within a graphics file

- $embeddedFrame configure (args)
- $embeddedFrame configuredSubFrames (frameConfiguration)
- $embeddedFrame instances ()
- $embeddedFrame subFrames ()

Table 2-7. embeddedFrame object commands

Command	Description
$embeddedFrame configure (args)	With no arguments lists the names and values of all options. With one argument lists the value of the specified option. **Arguments** **class** : identifies the object as an embeddedFrame **name** :
$embeddedFrame configuredSubFrames (frameConfiguration)	Obtains the sub-frames. **Argument** **frameConfiguration** :defines a level in a VHDL configuration declaration from which the bindings are selected **Returns** A list of sub-frame information, each consecutive pair represents a frame and a frameConfiguration in that order.
$embeddedFrame instances ()	Obtains the component instances, refer to "instance object commands" on page 122, directly contained within the frame. Instances contained within a sub-frame are not included. **Returns** A list of instances.
$embeddedFrame subFrames ()	Obtains the sub-frames. **Returns** A list of sub-frames.

entity option object commands

Associated with the interface to a design unit. Corresponds to a VHDL entity, a Verilog module and an HDS symbol or block interface file

- $entity architecture (name)
- $entity configure (args)
- $entity file ()
- $entity packages ()

Table 2-8. entity option object commands

Command	Description
$entity architecture (name)	Obtains the architecture for an entity. **Argument** name (= {}) The name of the architecture to find, if omitted uses the default view settings to choose between alternative architectures. **Returns** The matching architecture, refer to “architecture object commands” on page 106.
$entity configure (args)	With no arguments lists the names and values of all options. With one argument lists the value of the specified option. **Arguments** **class** : identifies the object as an entity **name** : name of the declaration itself **fullName** : fully qualified name including those of parent objects **startLine** : the line number where the declaration begins in the source file **refName** : Name that can be to refer to the object when running tasks or accessing properties
$entity file ()	Obtains the file containing the object. **Returns:** A single object, refer to “file object commands” on page 116.

Table 2-8. entity option object commands (cont.)

Command	Description
$entity packages ()	Obtains the packages referenced. **Returns** A list of packages, see "packageHeader object commands" on page 126, or an empty list if the declaration is not VHDL.

file object commands

- $file configure (args)
- $file declarations ()
- $file generated ()
- $file includes ()
- $file library ()

Table 2-9. file object commands

Command	Description
$file configure (args)	With no arguments lists the names and values of all options. With one argument lists the value of the specified option. **Arguments** **relativePathname** : pathname relative to the library mapping **language** : HDL language version : version checked out from the version management system **timestamp** : time the file was last modified type : identifies the type of the file **exclusionType** : **designDir** : The full hard pathname of the design side data directory **userDir** : The full hard pathname of the design side data directory **refName** : Name that can be to refer to the object when running tasks or accessing properties **class** : identifies the object as a file
$file declarations ()	Obtains the declarations in the file. **Returns** A list of declaration objects.
$file generated ()	Gets the HDL file generated from the current file. **Returns** The file.

Table 2-9. file object commands (cont.)

Command	Description
$file includes ()	Gets the include files. **Returns** A list of files or the empty list of the language is not Verilog.
$file library ()	Gets the library containing the file. **Returns** The library.

forFrame object commands

A VHDL 'for generate' statement.

- $forFrame configure (args)
- $forFrame configuredSubFrames (frameConfiguration)
- $forFrame instances ()
- $forFrame subFrames ()

Table 2-10. forFrame object commands

Command	Description
$forFrame configure (args)	With no arguments lists the names and values of all options. With one argument lists the value of the specified option. **Arguments** **leftBound** : **rightBound** : **class** : identifies the object as a forFrame **name** :
$forFrame configuredSubFrames (frameConfiguration)	Obtains the sub-frames. **Arguments** **frameConfiguration**: defines a level in a VHDL configuration declaration from which the bindings are selected **Returns** A list of sub-frame information, each consecutive pair represents a frame and a frameConfiguration in that order.
$forFrame instances ()	Obtains the component instances, refer to "instance object commands" on page 122, directly contained within the frame. Instances contained within a sub-frame are not included. **Returns** A list of instances.
$forFrame subFrames ()	Obtains the sub-frames. **Returns** A list of sub-frames.

frameConfiguration object commands

Represents a single level within the hierarchical structure of a VHDL configuration declaration, see configurationRefApi.tcl. These objects have do little in themselves but are passed as arguments to control the path through the design hierarchy. Refer to configuredChild in “instance object commands” on page 122 and configuredSubFrames in“architecture object commands” on page 106.

$frameConfiguration configure (args)

Table 2-11. frameConfiguration object commands

Command	Description
$frameConfiguration configure (args)	With no arguments lists the names and values of all options. With one argument lists the value of the specified option. **Arguments:** **name** : **class** : identifies the object as a frameConfiguration

frame object commands

- $frame configure (args)
- $frame configuredSubFrames (frameConfiguration)
- $frame instances ()
- $frame subFrames ()

Table 2-12. frame object commands

Command	Description
$frame configure (args)	With no arguments lists the names and values of all options. With one argument lists the value of the specified option. **Arguments** **name** : **class** : identifies the object as a frame
$frame configuredSubFrames (frameConfiguration)	Obtains the sub-frames. **Argument** **frameConfiguration**: defines a level in a VHDL configuration declaration from which the bindings are selected **Returns** A list of sub-frame information, each consecutive pair represents a frame and a frameConfiguration in that order.
$frame instances ()	Obtains the component instances, see “instance object commands” on page 122, directly contained within the frame. Instances contained within a sub-frame are not included. **Returns** A list of instances.
$frame subFrames ()	Obtains the sub-frames. **Returns** A list of sub-frames.

ifFrame object commands

A VHDL 'if generate' statement or a Verilog 'ifdef

- $ifFrame configure (args)
- $ifFrame configuredSubFrames (frameConfiguration)
- $ifFrame instances ()
- $ifFrame subFrames ()

Table 2-13. ifFrame object commands

Command	Description
$ifFrame configure (args)	With no arguments lists the names and values of all options. With one argument lists the value of the specified option. **Arguments** **class** : identifies the object as an ifFrame **name** :
$ifFrame configuredSubFrames (frameConfiguration)	Obtains the sub-frames. **Argument** **frameConfiguration**: defines a level in a VHDL configuration declaration from which the bindings are selected **Returns** A list of sub-frame information, each consecutive pair represents a frame and a frameConfiguration in that order.
$ifFrame instances ()	Obtains the component instances, see“instance object commands” on page 122, directly contained within the frame. Instances contained within a sub-frame are not included. **Returns** A list of instances.
$ifFrame subFrames ()	Obtains the sub-frames. **Returns** A list of sub-frames.

instance object commands

A component instance in a block diagram, IBD or structural HDL text file.

- $instance child ()
- $instance configure (args)
- $instance configuredChild (frameConfiguration)

Table 2-14. instance object commands

Command	Description
$instance child ()	Obtains the architecture under the component instance. **Returns** The architecture object.
$instance configure (args)	With no arguments lists the names and values of all options. With one argument lists the value of the specified option. **Arguments** **name** : **class** : identifies the object as an instance
$instance configuredChild (frameConfiguration)	Like 'child' but supporting VHDL configuration declarations. **Arguments** **frameConfiguration** see “frameConfiguration object commands” on page 119. **Returns** A 3 element list containing: The configuration declaration object, if specified in the component binding; see “configuration object commands” on page 109. The architecture object, see“architecture object commands” on page 106. The frame configuration for the root of the child architecture.

library object commands

A library object is obtained from the 'library open' command. It provides access to the files and declarations within the library.

- $library close ()
- $library configure (args)
- $library declaration (primaryName secondaryName)
- $library file (relativePathname)
- $library files ()
- $library hdlDirectory ()

Table 2-15. library object commands

Command	Description
$library close ()	Removes the library from memory. It is an error to close a library while an editor window or Design Explorer tab is showing design items in the library. **Returns** None.
$library configure (args)	With no arguments lists the names and values of all options. With one argument lists the value of the specified option. **Arguments** **name** : name of the library **type** : regular, standard or downstreamOnly **hdlDir** : directory as literally given **hdsDir** : directory as literally given **hardHdlDir** : directory with variables expanded **hardHdsDir** : directory with variables expanded **refName** : Name that can be to refer to the object when running tasks or accessing properties **class** : identifies the object as a library

Table 2-15. library object commands (cont.)

Command	Description
$library declaration (primaryName secondaryName)	Finds an entity, architecture, package or configuration declaration by name. Raises an error if a matching architecture is not available. **Argument** **primaryName**:name to look for **secondaryName** (= {}) :only specified when looking for an architecture can be the empty string if the default architecture should be used **Returns** The matching declaration.
$library file (relativePathname)	Finds a file, raises an error if not available. **Argument** **relativePathname**:pathname of the file relative to the root directory of the library. **Returns** The file object.
$library files ()	Obtains the files of the library, see “file object commands” on page 116. **Returns** List of files.
$library hdlDirectory ()	Obtains the root HDL directory of the library. **Returns** Directory object.

machineFrame object commands

Represents hierarchy or concurrency within a state machine, algorithmic state machine or flow chart file.

- $machineFrame configure (args)
- $machineFrame configuredSubFrames (frameConfiguration)
- $machineFrame instances ()
- $machineFrame subFrames ()

Table 2-16. machineFrame object commands

Command	Description
$machineFrame configure (args)	With no arguments lists the names and values of all options. With one argument lists the value of the specified option. **Arguments** **class** : identifies the object as a machineFrame **name** :
$machineFrame configuredSubFrames (frameConfiguration)	Obtains the sub-frames. **Arguments** **frameConfiguration**: defines a level in a VHDL configuration declaration from which the bindings are selected **Returns** A list of sub-frame information, each consecutive pair represents a frame and a frameConfiguration in that order.
$machineFrame instances ()	Obtains the component instances, see“instance object commands” on page 122, directly contained within the frame. Instances contained within a sub-frame are not included. **Returns** A list of instances.
$machineFrame subFrames ()	Obtains the sub-frames. **Returns** A list of sub-frames.

packageHeader object commands

- $packageHeader body ()
- $packageHeader configure (args)
- $packageHeader file ()
- $packageHeader packages ()

Table 2-17. packageHeader object commands

Command	Description
$packageHeader body ()	Gets the package body implementing the interface of a package header, see "packageBody object commands" on page 127. **Returns** The body object.
$packageHeader configure (args)	With no arguments lists the names and values of all options. With one argument lists the value of the specified option. **Arguments** **class** : identifies the object as a packageHeader **name** : name of the declaration itself **fullName** : fully qualified name including those of parent objects **startLine** : the line number where the declaration begins in the source file **refName** : Name that can be to refer to the object when running tasks or accessing properties
$packageHeader file ()	Obtains the file containing the object. **Returns** A single object, see "file object commands" on page 116.
$packageHeader packages ()	Obtains the packages referenced. **Returns** A list of packages, see "packageHeader object commands" on page 126, or an empty list if the declaration is not VHDL.

packageBody object commands

A VHDL package body

- $packageBody configure (args)
- $packageBody file ()
- $packageBody packages ()

Table 2-18. packageBody object commands

Command	Description
$packageBody configure (args)	With no arguments lists the names and values of all options. With one argument lists the value of the specified option. **Arguments** **class** : identifies the object as a packageBody **name** : name of the declaration itself **fullName** : fully qualified name including those of parent objects **startLine** : the line number where the declaration begins in the source file **refName** : Name that can be to refer to the object when running tasks or accessing properties
$packageBody file ()	Obtains the file containing the object. **Returns** A single object, see "file object commands" on page 116.
$packageBody packages ()	Obtains the packages referenced. **Returns** A list of packages, see "packageHeader object commands" on page 126, or an empty list if the declaration is not VHDL.

Command List

Index

— Symbols —

— S —

— V —

End-User License Agreement

The latest version of the End-User License Agreement is available on-line at:
www.mentor.com/eula

IMPORTANT INFORMATION

USE OF ALL SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE PRODUCTS. USE OF SOFTWARE INDICATES CUSTOMER'S COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

END-USER LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Software (as defined in Section 2) and hardware (collectively "Products") between the company acquiring the Products ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, in the case of Software received electronically, certify destruction of Software and all accompanying items within five days after receipt of Software and receive a full refund of any license fee paid.

1. **ORDERS, FEES AND PAYMENT.**

   1.1. To the extent Customer (or if agreed by Mentor Graphics, Customer's appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement ("Order(s)"), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement, any applicable addenda and the applicable quotation, whether or not these documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order will not be effective unless agreed in writing by an authorized representative of Customer and Mentor Graphics.

   1.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will state separately in the applicable invoice(s). Unless timely provided with a valid certificate of exemption or other evidence that items are not taxable, Mentor Graphics will invoice Customer for all applicable taxes including, but not limited to, VAT, GST, sales tax and service tax. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer's sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer's behalf, Customer shall be liable for payment under Orders placed by such third party in the event of default.

   1.3. All Products are delivered FCA factory (Incoterms 2000), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics retains a security interest in all Products delivered under this Agreement, to secure payment of the purchase price of such Products, and Customer agrees to sign any documents that Mentor Graphics determines to be necessary or convenient for use in filing or perfecting such security interest. Mentor Graphics' delivery of Software by electronic means is subject to Customer's provision of both a primary and an alternate e-mail address.

2. **GRANT OF LICENSE.** The software installed, downloaded, or otherwise acquired by Customer under this Agreement, including any updates, modifications, revisions, copies, documentation and design data ("Software") are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form (except as provided in Subsection 5.2); (b) for Customer's internal business purposes; (c) for the term of the license; and (d) on the computer hardware and at the site authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Customer may have Software temporarily used by an employee for telecommuting purposes from locations other than a Customer office, such as the employee's residence, an airport or hotel, provided that such employee's primary place of employment is the site where the Software is authorized for use. Mentor Graphics' standard policies and programs, which vary depending on Software, license fees paid or services purchased, apply to the following: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be technically implemented through the use of authorization codes or similar devices); and (c) support services provided, including eligibility to receive telephone support, updates, modifications, and revisions. For the avoidance of doubt, if Customer requests any change or enhancement to Software, whether in the course of

receiving support or consulting services, evaluating Software, performing beta testing or otherwise, any inventions, product improvements, modifications or developments made by Mentor Graphics (at Mentor Graphics' sole discretion) will be the exclusive property of Mentor Graphics.

3. **ESC SOFTWARE.** If Customer purchases a license to use development or prototyping tools of Mentor Graphics' Embedded Software Channel ("ESC"), Mentor Graphics grants to Customer a nontransferable, nonexclusive license to reproduce and distribute executable files created using ESC compilers, including the ESC run-time libraries distributed with ESC C and C++ compiler Software that are linked into a composite program as an integral part of Customer's compiled computer program, provided that Customer distributes these files only in conjunction with Customer's compiled computer program. Mentor Graphics does NOT grant Customer any right to duplicate, incorporate or embed copies of Mentor Graphics' real-time operating systems or other embedded software products into Customer's products or applications without first signing or otherwise agreeing to a separate agreement with Mentor Graphics for such purpose.

4. **BETA CODE.**

4.1. Portions or all of certain Software may contain code for experimental testing and evaluation ("Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and Customer's use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form.

4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.

4.3. Customer agrees to maintain Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer's feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 4.3 shall survive termination of this Agreement.

5. **RESTRICTIONS ON USE.**

5.1. Customer may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Customer shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. Customer shall not make Products available in any form to any person other than Customer's employees and on-site contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use it except as permitted by this Agreement. Customer shall give Mentor Graphics written notice of any unauthorized disclosure or use of the Products as soon as Customer learns or becomes aware of such unauthorized disclosure or use. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive any source code from Software. Log files, data files, rule files and script files generated by or for the Software (collectively "Files"), including without limitation files containing Standard Verification Rule Format ("SVRF") and Tcl Verification Format ("TVF") which are Mentor Graphics' proprietary syntaxes for expressing process rules, constitute or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Customer may use Files containing SVRF or TVF only with Mentor Graphics products. Under no circumstances shall Customer use Software or Files or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Software, or disclose to any third party the results of, or information pertaining to, any benchmark.

5.2. If any Software or portions thereof are provided in source code form, Customer will use the source code only to correct software errors and enhance or modify the Software for the authorized use. Customer shall not disclose or permit disclosure of source code, in whole or in part, including any of its methods or concepts, to anyone except Customer's employees or contractors, excluding Mentor Graphics competitors, with a need to know. Customer shall not copy or compile source code in any manner except to support this authorized use.

5.3. Customer may not assign this Agreement or the rights and duties under it, or relocate, sublicense or otherwise transfer the Products, whether by operation of law or otherwise ("Attempted Transfer"), without Mentor Graphics' prior written consent and payment of Mentor Graphics' then-current applicable relocation and/or transfer fees. Any Attempted Transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and/or the licenses granted under this Agreement. The terms

of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer's permitted successors in interest and assigns.

5.4. The provisions of this Section 5 shall survive the termination of this Agreement.

6. **SUPPORT SERVICES.** To the extent Customer purchases support services, Mentor Graphics will provide Customer updates and technical support for the Products, at the Customer site(s) for which support is purchased, in accordance with Mentor Graphics' then current End-User Support Terms located at http://supportnet.mentor.com/about/legal/.

7. **AUTOMATIC CHECK FOR UPDATES; PRIVACY.** Technological measures in Software may communicate with servers of Mentor Graphics or its contractors for the purpose of checking for and notifying the user of updates and to ensure that the Software in use is licensed in compliance with this Agreement. Mentor Graphics will not collect any personally identifiable data in this process and will not disclose any data collected to any third party without the prior written consent of Customer, except to Mentor Graphics' outside attorneys or as may be required by a court of competent jurisdiction.

8. **LIMITED WARRANTY.**

8.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Products, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Products will meet Customer's requirements or that operation of Products will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Software under an Order and does not renew or reset, for example, with the delivery of (a) Software updates or (b) authorization codes or alternate Software under a transaction involving Software re-mix. This warranty shall not be valid if Products have been subject to misuse, unauthorized modification or improper installation. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF THE PRODUCTS TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF THE PRODUCTS THAT DO NOT MEET THIS LIMITED WARRANTY, PROVIDED CUSTOMER HAS OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) PRODUCTS PROVIDED AT NO CHARGE; OR (C) BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."

8.2. THE WARRANTIES SET FORTH IN THIS SECTION 8 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO PRODUCTS PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

9. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS' OR ITS LICENSORS' LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT RECEIVED FROM CUSTOMER FOR THE HARDWARE, SOFTWARE LICENSE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

10. **HAZARDOUS APPLICATIONS.** CUSTOMER ACKNOWLEDGES IT IS SOLELY RESPONSIBLE FOR TESTING ITS PRODUCTS USED IN APPLICATIONS WHERE THE FAILURE OR INACCURACY OF ITS PRODUCTS MIGHT RESULT IN DEATH OR PERSONAL INJURY ("HAZARDOUS APPLICATIONS"). NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF MENTOR GRAPHICS PRODUCTS IN OR FOR HAZARDOUS APPLICATIONS. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

11. **INDEMNIFICATION.** CUSTOMER AGREES TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE OR LIABILITY, INCLUDING ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH THE USE OF PRODUCTS AS DESCRIBED IN SECTION 10. THE PROVISIONS OF THIS SECTION 11 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

12. **INFRINGEMENT.**

12.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product acquired by Customer hereunder infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay costs and damages finally awarded against Customer that are attributable to the action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance

to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

12.2. If a claim is made under Subsection 12.1 Mentor Graphics may, at its option and expense, (a) replace or modify the Product so that it becomes noninfringing; (b) procure for Customer the right to continue using the Product; or (c) require the return of the Product and refund to Customer any purchase price or license fee paid, less a reasonable allowance for use.

12.3. Mentor Graphics has no liability to Customer if the action is based upon: (a) the combination of Software or hardware with any product not furnished by Mentor Graphics; (b) the modification of the Product other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of the Product as part of an infringing process; (e) a product that Customer makes, uses, or sells; (f) any Beta Code or Product provided at no charge; (g) any software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; or (h) infringement by Customer that is deemed willful. In the case of (h), Customer shall reimburse Mentor Graphics for its reasonable attorney fees and other costs related to the action.

12.4. THIS SECTION 12 IS SUBJECT TO SECTION 9 ABOVE AND STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS FOR DEFENSE, SETTLEMENT AND DAMAGES, AND CUSTOMER'S SOLE AND EXCLUSIVE REMEDY, WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT PROVIDED UNDER THIS AGREEMENT.

13. **TERMINATION AND EFFECT OF TERMINATION.** If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized term.

13.1. Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement immediately upon written notice if Customer: (a) exceeds the scope of the license or otherwise fails to comply with the licensing or confidentiality provisions of this Agreement, or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. For any other material breach of any provision of this Agreement, Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement upon 30 days written notice if Customer fails to cure the breach within the 30 day notice period. Termination of this Agreement or any license granted hereunder will not affect Customer's obligation to pay for Products shipped or licenses granted prior to the termination, which amounts shall be payable immediately upon the date of termination.

13.2. Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination, Customer shall ensure that all use of the affected Products ceases, and shall return hardware and either return to Mentor Graphics or destroy Software in Customer's possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any of the affected Products or copies of Software in any form.

14. **EXPORT.** The Products provided hereunder are subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain products and information about the products to certain countries and certain persons. Customer agrees that it will not export Products in any manner without first obtaining all necessary approval from appropriate local and United States government agencies.

15. **U.S. GOVERNMENT LICENSE RIGHTS.** Software was developed entirely at private expense. All Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to US FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in this Agreement, except for provisions which are contrary to applicable mandatory federal laws.

16. **THIRD PARTY BENEFICIARY.** Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, Microsoft Corporation and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.

17. **REVIEW OF LICENSE USAGE.** Customer will monitor the access to and use of Software. With prior written notice and during Customer's normal business hours, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system and records deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FLEXlm or FLEXnet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. The provisions of this Section 17 shall survive the termination of this Agreement.

18. **CONTROLLING LAW, JURISDICTION AND DISPUTE RESOLUTION.** The owners of certain Mentor Graphics intellectual property licensed under this Agreement are located in Ireland and the United States. To promote consistency around the world, disputes shall be resolved as follows: excluding conflict of laws rules, this Agreement shall be governed by and construed under the laws of the State of Oregon, USA, if Customer is located in North or South America, and the laws of Ireland if Customer is located outside of North or South America. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of the courts of Portland, Oregon when the laws of Oregon apply, or Dublin, Ireland when the laws of Ireland apply. Notwithstanding the foregoing, all disputes in Asia arising out of or in relation to this Agreement shall be resolved by arbitration in Singapore before a single arbitrator to be appointed by the chairman of the Singapore International

Arbitration Centre ("SIAC") to be conducted in the English language, in accordance with the Arbitration Rules of the SIAC in effect at the time of the dispute, which rules are deemed to be incorporated by reference in this section. This section shall not restrict Mentor Graphics' right to bring an action against Customer in the jurisdiction where Customer's place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.

19. **SEVERABILITY.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.

20. **MISCELLANEOUS.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements, including but not limited to any purchase order terms and conditions. Some Software may contain code distributed under a third party license agreement that may provide additional rights to Customer. Please see the applicable Software documentation for details. This Agreement may only be modified in writing by authorized representatives of the parties. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.

Rev. 100615, Part No. 246066