

Virtuoso Hierarchy Editor User Guide

**Product Version IC23.1
August 2023**

© 2023 Cadence Design Systems, Inc.

Printed in the United States of America.

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission.

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 800.862.4522. All other trademarks are the property of their respective holders.

Restricted Permission: This publication is protected by copyright law and international treaties and contains trade secrets and proprietary information owned by Cadence. Unauthorized reproduction or distribution of this publication, or any portion of it, may result in civil and criminal penalties. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. Unless otherwise agreed to by Cadence in writing, this statement grants Cadence customers permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used only in accordance with a written agreement between Cadence and its customer.
2. The publication may not be modified in any way.
3. Any authorized copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement.
4. The information contained in this document cannot be used in the development of like products or software, whether for internal or external use, and shall not be used for the benefit of any other party, whether or not for consideration.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Cadence is committed to using respectful language in our code and communications. We are also active in the removal and replacement of inappropriate language from existing content. This product documentation may however contain material that is no longer considered appropriate but still reflects long-standing industry terminology. Such content will be addressed at a time when the related software can be updated without end-user impact.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor

Contents

1

<u>Virtuoso Hierarchy Editor Overview</u>	1
<u>Starting the Hierarchy Editor</u>	2
<u>Virtuoso Hierarchy Editor Command-line Options</u>	4
<u>Hierarchy Editor Menu Commands</u>	8
<u>Hierarchy Editor Toolbar</u>	11
<u>Search Functionality of Virtuoso Hierarchy Editor</u>	12
<u>Search Mechanism in the Hierarchy Editor toolbar</u>	12
<u>Search Mechanism in the Search Assistant Pane</u>	13
<u>Using the Top Cell and Global Bindings Section in Hierarchy Editor</u>	15
<u>Displaying the Table View in Hierarchy Editor</u>	17
<u>Cell Bindings Table in the Table View Tab</u>	18
<u>Mark As External HDL Text (AMS UNL only)</u>	18
<u>Cell Bindings Table Context-Sensitive Menu</u>	19
<u>Multiple Selection and Action</u>	19
<u>Performing Multiple Selection Using Cell Bindings Table</u>	20
<u>Instance Bindings Table in the Table View Tab</u>	22
<u>Displaying the Tree View in Hierarchy Editor</u>	24
<u>Displaying the Status Bar and Window Border in Hierarchy Editor</u>	25
<u>Customizing Hierarchy Editor Menus</u>	26
<u>Customizing Hierarchy Editor Columns</u>	28
<u>Specifying the Columns to Display</u>	28
<u>Checking for unused configurations</u>	29
<u>Changing Column Order</u>	30
<u>Resizing Columns</u>	30
<u>Sorting Data</u>	31
<u>Updating the Configuration Automatically</u>	32
<u>Filtering Cellviews</u>	33
<u>Viewing the Legend</u>	35
<u>Changing the Hierarchy Editor Fonts</u>	37

Virtuoso Hierarchy Editor User Guide

<u>Exiting the Hierarchy Editor</u>	39
<u>Saving Custom Settings</u>	39

2

<u>Design Hierarchy Configuration View</u>	41
<u>Opening Configurations in Hierarchy Editor</u>	43
<u>Creating a New Configuration</u>	45
<u>Editing Configurations in Hierarchy Editor</u>	47
<u>Editing the Cell Bindings and Instance Bindings Tables</u>	48
<u>Adding Views to Cell or Instance Bindings</u>	48
<u>Deleting Views from Cell or Instance Bindings</u>	49
<u>Viewing Instances Contained by Other Instances</u>	49
<u>Performing a Global Change on a Group of Instances</u>	49
<u>Viewing All Instances of a Subtree</u>	50
<u>Hierarchy Editor Sub-Configurations</u>	52
<u>Viewing the Description File</u>	53
<u>Verifying Binding Rules</u>	54
<u>Working with Templates in Hierarchy Editor</u>	56
<u>Creating Templates</u>	56
<u>Using Templates</u>	57
<u>Copying a New Template to the Predefined Templates Location</u>	58
<u>Saving Configurations</u>	59
<u>Saving a Configuration</u>	59
<u>Saving a Configuration as VHDL</u>	60
<u>Saving a Configuration as Verilog</u>	61
<u>Comparing Configurations</u>	64
<u>Comparing Configurations from the Command-Line</u>	64
<u>Traversing Configurations</u>	67
<u>Traversing Configurations from the Command-Line</u>	67
<u>Using the Virtuoso Schematic Editor with the Hierarchy Editor</u>	69
<u>Update Process for Read-Only Libraries in pc.db Files</u>	71
<u>Creating a Snapshot Configuration</u>	72
<u>Checking a Snapshot Configuration</u>	73
<u>Clearing Snapshot Bindings</u>	74

3

Design Components in Hierarchy Editor	77
Global Bindings	77
Cell Bindings	78
Instance Bindings	78
Occurrence Bindings	79
Define a Library for an Object	79
Define a View for an Object	80
Rules Definition at the Global Level	81
Defining a View List at the Global Level	82
Creating a Global View List	83
Deleting Views from the Global View List	83
Defining a Library List at the Global Level	85
Creating a Library List at the Global Level	85
Defining a Stop List at the Global Level	87
Defining a Constraint List at the Global Level	89
Rules Definition at the Cell Level	91
Changing Library Bindings on a Per-Cell Basis	92
Creating a Cell-Level Library List	92
Changing View Bindings on a Per-Cell Basis	94
Specifying Cell Binding for a Cell and the Related Objects	94
Defining Stop Points on a Per-Cell Basis	96
Viewing the Cell Instantiations	96
Defining Bind-to-Open on a Per Cell Basis	97
Viewing Instantiations	98
Rules Definition at the Instance Level	99
Changing View Bindings on a Per-Instance Basis	100
Changing Library Bindings on a Per-Instance Basis	103
Defining Stop Points on a Per-Instance Basis	106
Defining Bind-to-Open on a Per-Instance Basis	108
Changing Instance Bindings Inside a Block	110
Rules Definition at the Occurrence Level	111
Difference between Occurrences and Instances	113
Defining Occurrence Bindings	115
Creating an Occurrence Binding	115

Virtuoso Hierarchy Editor User Guide

<u>Editing an Occurrence Binding</u>	116
<u>Removing an Occurrence Binding</u>	118
<u>Defining Occurrence Stop Points</u>	119
<u>Adding an Occurrence Stop Point</u>	119
<u>Removing an Occurrence Stop Point</u>	119
<u>Defining Occurrence-Level Bind-to-Open</u>	121
<u>Setting Bind-to-Open on an Occurrence</u>	121
<u>Removing Occurrence-Level Bind-to-Open</u>	122
<u>Using Text Files in Configuration</u>	123
<u>Using Source Files in Configuration</u>	123
<u>Using Verilog Files in Configuration</u>	125
<u>Compiler Options</u>	126
<u>Referencing a Verilog File</u>	126
<u>Creating and Editing Constants</u>	129
<u>Changing the Views in the View Choices List Box</u>	131
<u>Wildcards in a View List</u>	132
<u>Library Scoped Views in a View List</u>	133
<u>Changing Binding Data Color Definitions</u>	134
<u>Saving Cell Bindings Table Data to a Text File</u>	136

4

<u>Setting Simulation-Control Properties</u>	139
<u>Displaying Properties in Hierarchy Editor</u>	140
<u>Properties Setting on the Design</u>	143
<u>Setting Properties on a Global Basis</u>	144
<u>Setting Properties on a Cell</u>	146
<u>Viewing Instantiations</u>	146
<u>Setting Properties on an Instance</u>	148
<u>Setting Instance Properties from the Tree View</u>	148
<u>Setting Instance Properties from the Table View</u>	149
<u>Setting Properties on an Occurrence</u>	151
<u>Removing Properties</u>	153
<u>Removing Properties from Specific Objects</u>	153
<u>Removing Properties from the Entire Design Configuration</u>	153
<u>Adding Property Columns</u>	155

5

Hierarchy Editor Plug-Ins	157
Loading Plug-In Applications	158
Loading a Plug-In from the Hierarchy Editor User Interface	158
Loading a Plug-in from the Command Line	158
Checking Imported Packages	160
Pin Checker	161
Performing a Global Pin Check	163
Performing a Single Instance Pin Check	164
Performing Multiple Instances Pin Check	165
Performing a Cell Instantiation Pin Check	167
Multiple Cell Instantiations Pin Check	168
Log Files Creation	170
Removing Plug-In Applications	171

A

Hierarchy Editor Environment Variables	173
List of Hierarchy Editor Environment Variables	173
setAmsSimulator	175
showInfo	176
bindingError	177
defaultBinding	178
highlight	179
notInUse	180
rowSelection	181
userBinding	182
maskLayoutStopLimit	183
viewChoices	184
maskLayoutStopLimit	185
showConstInViewList	186
showInfo	187
filter	188
name	189
autoGenPcdbFiles	190

Virtuoso Hierarchy Editor User Guide

<u>useNCCompilers</u>	191
<u>Compare Configuration Variables</u>	192
<u>outputDir</u>	193
<u>spaceIndent</u>	194
<u>minTraversal</u>	195
<u>subConfigs</u>	196
<u>diffTool</u>	197
<u>Pin Check Variables</u>	198
<u>pcLogFileOpenMode</u>	199
<u>pcLogFilePath</u>	200

B

<u>Hierarchy Editor Forms</u>	201
<u>Add Occurrence Binding Form</u>	203
<u>Compare Config Form</u>	204
<u>Edit Constants Form</u>	207
<u>Edit Description Form</u>	208
<u>Explain Form (Selected Cell)</u>	209
<u>Explain Form (Selected Instance)</u>	210
<u>Explain Form (Selected Occurrence)</u>	211
<u>Filters Form</u>	212
<u>New Configuration Form</u>	214
<u>Save As Form</u>	216
<u>Save As Verilog Form</u>	217
<u>Save As VHDL Form</u>	218
<u>Open Form</u>	219
<u>Options Form (Cell Table)</u>	220
<u>Options Form (Instance Table)</u>	221
<u>Options Form (Tree)</u>	222
<u>Options Form (General)</u>	223
<u>Options Form (Checks)</u>	225
<u>Package Checking Viewer Form</u>	226
<u>Table View</u>	226
<u>Tree View</u>	226
<u>Pin Checker Options Form</u>	228

Virtuoso Hierarchy Editor User Guide

<u>Populate Library with Verilog Modules Form</u>	230
<u>Reference Verilog Modules Form</u>	231
<u>Traverse config Form</u>	233
<u>Use Template Form</u>	235
<u>View List Building Forms</u>	236
<u>Virtuoso Hierarchy Editor Editing Form</u>	237
<u>Virtuoso Hierarchy Editor Editing Form (Table View)</u>	239
<u>Virtuoso Hierarchy Editor Editing Form (Tree View)</u>	240

C

<u>Hierarchy Editor SKILL Functions</u>	241
<u>hedRegUICustomFunc</u>	242
<u>hnlRunPinChecker</u>	243

Virtuoso Hierarchy Editor User Guide

Virtuoso Hierarchy Editor Overview

The Virtuoso Hierarchy Editor lets you view many levels of a single design using either a table view or a tree view. The Hierarchy Editor also lets you create a configuration that provides expansion information for mixed-signal partitioning. The configuration file is the `expand.cfg` file, which specifies how a design is to be expanded.

You can use the Hierarchy Editor to:

- Display the configuration in order to traverse the hierarchy of your design
- Display default cell and instance bindings
- Change global library, view, and stop lists
- Change cell and instance bindings
- Change the inherited view list and library list for cells and instances
- Specify occurrence bindings
- Create new configurations

Related Topics

[Starting the Hierarchy Editor](#)

Starting the Hierarchy Editor

You can start the Hierarchy Editor from a terminal window, from the Virtuoso Studio design environment, and from Cadence applications such as the Virtuoso schematic editor.

To start the Hierarchy Editor from a terminal window,

- ➔ Type the following command:

```
cdsHierEditor &
```

For a list of options with which you can start the Hierarchy Editor, see [Filters Form](#).

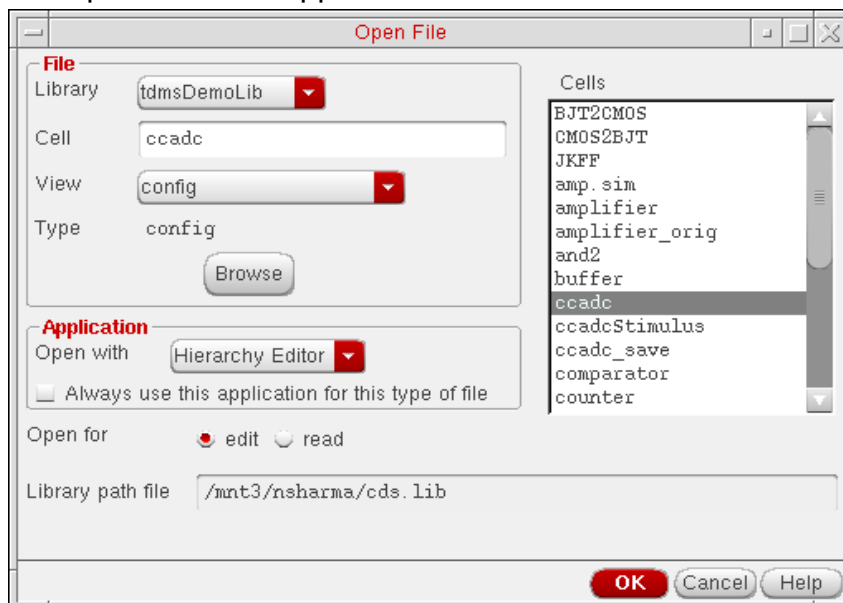
To start the Hierarchy Editor from the Virtuoso Studio design environment,

1. Start the Virtuoso Studio design environment.

```
virtuoso &
```

2. Choose *File – Open*.

The Open File form appears.

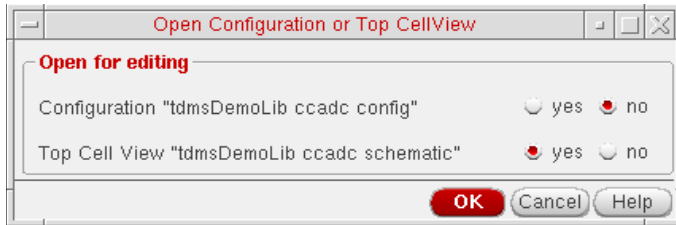


3. In the *File* section of the form, select the library, cell, and view of the configuration that you want to open.
4. In the *Application* field, set the application to *Hierarchy Editor*, if it is not already set.
5. In the *Open for* field, select *edit* if you want to edit the configuration or *read* if you want to open the configuration read-only.
6. Click *OK*.

Virtuoso Hierarchy Editor User Guide

Virtuoso Hierarchy Editor Overview

The Open Configuration or Top Cellview form appears.



7. In the form, in the *Configuration “configName”* field, select *yes*. This selection opens the Hierarchy Editor.
8. In the *Top Cell View “top cellview Name”* field, select *yes* if you want to open the schematic editor also or *no* if you do not want to open the schematic editor.
9. Click *OK*.

The Hierarchy Editor opens and displays the configuration. If you chose *yes* in the *Top Cell View* field, the schematic editor also opens and displays the top cellview of the configuration. You can start the Hierarchy Editor from other Cadence applications such as the schematic editor.

Related Topics

[Open Form](#)

[Virtuoso Hierarchy Editor Command-line Options](#)

[Hierarchy Editor Toolbar](#)

[Exiting the Hierarchy Editor](#)

Virtuoso Hierarchy Editor Command-line Options

When you start the Hierarchy Editor from the command line, you can specify the following options.

```
cdsHierEditor
  [-h[elp]]
  [-cdslib filePath]
  [-lib libname]
  [-cell cellname]
  [-view viewname]
  [-mode {r|w|a}]
  [-namespace nmpName]
  [-cdslib filePath]
  [-log filename]
  [-restore dirname]
  [-tree]
  [-version]
  [-ignoreRootConfig]
  [-plugin pluginName [pluginOptions ...]]
```

If you use the `-plugin` option, you must specify it as the last argument. The Hierarchy Editor ignores all arguments after `-plugin`, except another `-plugin` argument.

Arguments

<code>-h[elp]</code>	Prints a list and a brief description of the command-line options.
<code>-cdslib <i>filePath</i></code>	Specifies the <code>cds.lib</code> file to load. The <code>cds.lib</code> file defines the libraries that you can use. If you do not specify this option, the Cadence Search File mechanism (CSF) is used to find the <code>cds.lib</code> file to load.
<code>-lib <i>libname</i></code>	Specifies the library name of the configuration to be opened.
<code>-cell <i>cellname</i></code>	Specifies the cell name of the configuration to be opened.
<code>-view <i>viewname</i></code>	Specifies the view name of the configuration to be opened. You must specify all three parts of the <code>-lib</code> , <code>-cell</code> , and <code>-view</code> options for the command to be complete.

Virtuoso Hierarchy Editor User Guide

Virtuoso Hierarchy Editor Overview

<code>-mode {r w a}</code>	<p>Specifies the editing mode (read, write, or append) in which the Hierarchy Editor opens the configuration. The default mode is append.</p> <p>The <code>-mode w</code> (write) option erases all data in the configuration file you specify and opens the empty configuration file.</p>
<code>-namespace <i>nmpName</i></code>	<p>Specifies the name space to operate in, for example, Verilog, or VHDL. All names are then displayed in that name space, regardless of the source description that is used.</p>
<code>-cdslib <i>filepath</i></code>	<p>Specifies the <code>cds.lib</code> file to load. The <code>cds.lib</code> file specifies the libraries to use.</p>
<code>-log <i>filename</i></code>	<p>By default the Hierarchy Editor creates a log file called <code>hierEditor.log</code> to record commands and messages from the session. If that file is locked, it creates <code>hierEditor.log.1</code>, and if that is also locked, <code>hierEditor.log.2</code>, and so on, until <code>hierEditor.log.10</code>.</p> <p>Use the <code>-log <i>filename</i></code> option if you want to specify another log file name and path.</p>
<code>-restoreDir <i>dirname</i></code>	<p>Loads the <code>hed.env</code> file from the specified directory after other Hierarchy Editor files (found with the Cadence Search File mechanism) have been loaded. The <code>hed.env</code> file contains saved environment settings.</p>
<code>-tree</code>	<p>Specifies that the tree view is to be displayed at startup. If this argument is not specified, the table view is displayed.</p>

Virtuoso Hierarchy Editor User Guide

Virtuoso Hierarchy Editor Overview

`-version`

Prints version information and exits. The `-version` option displays the following version information:

- Version number of the Hierarchy Editor
- The versions of input files the Hierarchy Editor supports. Examples of input files include `expand.cfg` and `pc.db`.
- The version of output files the Hierarchy Editor outputs. Examples of output files include `expand.cfg` and Verilog.

For example:

```
Tool:    cdsHierEditor    05.01.000-b005
Input:   expand.cfg       04.04.003
Input:   expand.cfg       05.00.000
Input:   pc.db           01.00
Output:  expand.cfg       05.00.000
Output:  Verilog         1364-1995
Output:  VHDL            1076-1993
```

`-ignoreRootConfig`

Ignores the root configuration in select and highlight messages between the Hierarchy Editor and Virtuoso® Design Environment applications.

By default, if you are not editing the same root configuration in your Virtuoso application and the Hierarchy Editor, select and highlight messages between the two applications are ignored. If you set the `-ignoreRootConfig` option, the Hierarchy Editor ignores the root configuration.

`-plugin pluginName`
`[pluginOptions ...]`

Loads the specified plug-in. You can specify command-line options for the plug-in. To load more than one plug-in, type `-plugin` before each plug-in name.

If you use the `-plugin` option, you must specify it as the last argument. The Hierarchy Editor ignores all arguments after `-plugin`, except another `-plugin` argument.

Plug-ins must be installed in the `your_install_dir/share/cdssetup/hierEditor/plugins` directory.

Virtuoso Hierarchy Editor User Guide

Virtuoso Hierarchy Editor Overview

Related Topics

[Hierarchy Editor Plug-Ins](#)

[Starting the Hierarchy Editor](#)

[Hierarchy Editor Toolbar](#)

Hierarchy Editor Menu Commands

Use these menu commands to operate the Hierarchy Editor.

Menu	Description
Launch	Opens the selected window.
<i>ADE L</i>	Opens the ADE L window.
<i>ADE Explorer</i>	Opens an existing ADE Explorer view or creates a new ADE Explorer view.
<i>Plugins</i>	Includes plugins that are currently registered with Hierarchy Editor. Selecting an option from this menu loads the plugin. This submenu is not displayed if the plugins are not available.
File	Selects action to take for the current configuration.
<i>New Config</i>	Creates a new configuration.
<i>Open</i>	Opens an existing configuration for editing.
<i>Open (Read-Only)</i>	Opens a configuration for viewing.
<i>Save</i>	Saves the open configuration.
<i>Save As</i>	Saves the open configuration under a new name.
<i>Save As VHDL</i>	Saves the open configuration as a VHDL file.
<i>Save As Verilog</i>	Saves the open configuration as a Verilog file.
<i>Populate Library</i>	Brings Verilog modules into a library.
<i>Compare Configs</i>	Compares two configurations and highlight their differences by opening the resulting design hierarchy side-by-side in a tree view.
<i>Traverse Config</i>	Traverses through a configuration and save the traversal results in an output file.
<i>Save Cell Table Data</i>	Saves data from the cell table into a text file.
<i>Save Defaults</i>	Saves your settings.
Edit	Edits the configuration.
<i>Undo</i>	Removes the last action. You can undo an unlimited number of actions.
<i>Redo</i>	Redoes the previously undone action.

Virtuoso Hierarchy Editor User Guide

Virtuoso Hierarchy Editor Overview

Menu	Description
<i>Constants</i>	Opens the Edit Constants form to let you create or edit constants.
<i>Description</i>	Opens the Edit Description form to let you edit the description of the configuration.
<i>Add Property Column</i>	Adds a column for a simulation-control property. This menu option is available only if the <i>View – Properties</i> option is selected.
<i>Remove Property Column</i>	Removes a property column that is currently displayed.
View	Sets the display options for the Hierarchy Editor. You can choose to display or hide the tree view or table view, instance table, properties, top cell section, global bindings section, and message area in the Hierarchy Editor window.
<i>Update</i>	<p>Recomputes the hierarchy based on the rules you entered. This command is unavailable until you have opened a configuration.</p> <p>You can also select the <i>Automatic Update</i> option, which is available in the Options form, to automatically update your configuration every time you make a change.</p>
<i>Tree</i>	Displays cell and instance bindings in a tree structure.
<i>Parts Table</i>	Displays cell and instance bindings in a table structure.
<i>Instance Table</i>	Displays the instance table. This option is only available when <i>View – Parts Table</i> is selected.
<i>Properties</i>	Displays property columns that let you set simulation-control properties. This menu option is available only if there is a property dictionary in your Cadence installation hierarchy.
<i>Top Cell</i>	Displays or hides the top cell information.
<i>Global Bindings</i>	Displays or hides the global library, view, and stop lists.
<i>Search</i>	Opens the interactive Search assistant pane that offers additional search viewing information compared to that offered by the HED toolbar.
<i>Recreate Full Hierarchy</i>	Recomputes the full hierarchy and recreates all <code>pc.db</code> files for any text views in the design.
<i>Filters</i>	Opens the Filters form to let you set or edit display filters.

Virtuoso Hierarchy Editor User Guide

Virtuoso Hierarchy Editor Overview

Menu	Description
<i>Options</i>	<p>Opens the Options form to let you select the columns you want to display in the Hierarchy Editor. The Options form also lets you set the <i>Automatic Update</i> option.</p> <p>You can save the display options to use when you restart the Hierarchy Editor.</p> <p>If you access the standalone version of the Hierarchy Editor, there is also the <i>View – Message Area</i> option. This is not however required in the Virtuoso-integrated version, where the Hierarchy Editor utilizes the CIW message area.</p>

Related Topics

[Virtuoso Hierarchy Editor Editing Form](#)

[Filters Form](#)

[Options Form \(Cell Table\)](#)

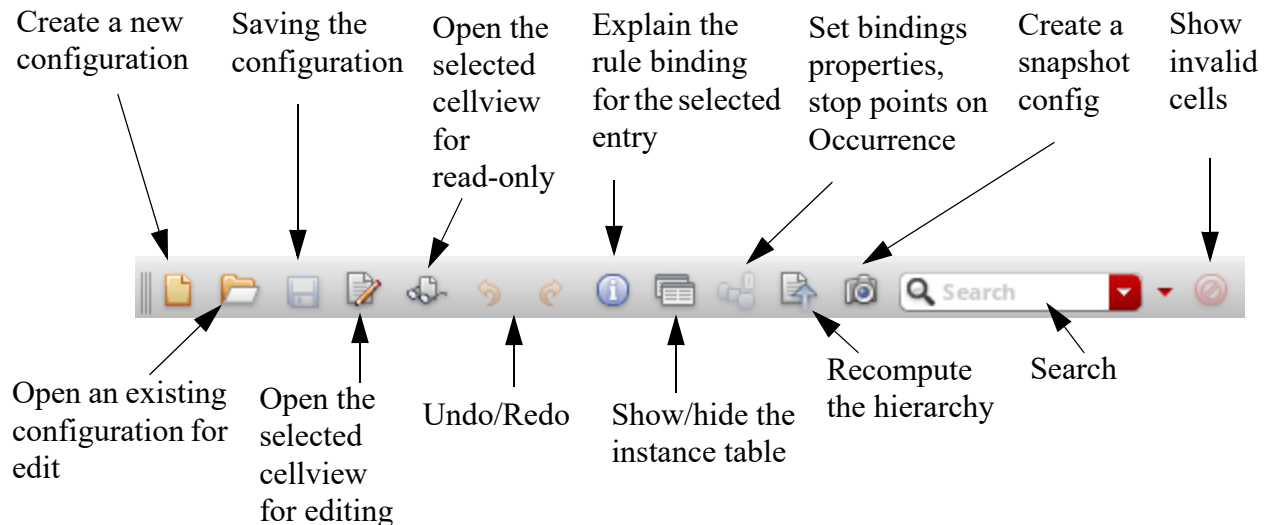
[Exiting the Hierarchy Editor](#)

[Edit Constants Form](#)

[Edit Description Form](#)

Hierarchy Editor Toolbar

You can access frequently used commands from the toolbar.



You can move the toolbar to any side of the main window. You cannot float it.

There is an *Invalid Cell* icon, which becomes active if there are any invalid (unbound) cells when a re-elaboration is done. If there are any unbound cells, clicking the icon updates the cell table to show those invalid cells at the top of the table.

Related Topics

[Starting the Hierarchy Editor](#)

[Hierarchy Editor Menu Commands](#)

[Virtuoso Hierarchy Editor Editing Form](#)




Search Functionality of Virtuoso Hierarchy Editor

The *Search* functionality in Hierarchy Editor comprises two search entry mechanisms:

- The HED Toolbar
- The Search Assistant Pane

Search Mechanism in the Hierarchy Editor toolbar

In the Hierarchy Editor toolbar these commands are available for the Search functionality.

Icon	Command	Description
	<i>Search text box</i>	Enters any search queries, such as the cell name, instance name, or occurrence path in the Virtuoso Hierarchy Editor window. If you type a string in the <i>Search</i> text box that begins with the same set of characters as those contained at the beginning of a search string entered earlier, the string entered earlier is automatically displayed as a choice (auto-complete). Once you type the search queries in the <i>Search</i> text box and press <i>Enter</i> , the results are displayed immediately in the <i>Show Results</i> drop-down list.
	<i>Show Results Drop-down List</i>	Displays a recent search history list where you can immediately recall the results of any recent searches and reinstate any of the search options selected when that search criteria was last applied. To clear the search history, you need to select the <i>Clear History</i> option from the <i>Show Results</i> drop-down list.
	<i>Advanced Search</i>	Refines your search results.

Virtuoso Hierarchy Editor User Guide

Virtuoso Hierarchy Editor Overview

Icon	Command	Description
	<i>Find With</i>	<p>Specifies the default matching operators. The options available here are AND (<i>All Of The Words</i>), OR (<i>Any Of The Words</i>), EXACTLY (<i>The Exact Phrase</i>), and NOT (<i>None Of The Words</i>)</p> <p>Default option: <i>All Of The Words</i></p> <p>For more information, see Query.</p>
	<i>Match Word</i>	<p>Specifies a regular query token that must be matched in the current content search data to display a successful result. The options available here are <i>Prefix</i>, <i>Substring</i>, <i>Exactly</i>, and <i>Suffix</i>.</p> <p>Default option: <i>Substring</i></p> <p>For more information, see Word.</p>
	<i>Using Case</i>	<p>Determines whether search results must be case sensitive (<i>Sensitive Match</i>) or whether any text case is acceptable (<i>Insensitive Match</i>).</p> <p>Default option: <i>Insensitive Match</i></p>

Search Mechanism in the Search Assistant Pane

The interactive *Search* assistant pane is a dockable/undockable pane that offers additional search viewing information compared to that offered by the HED toolbar.

The *Search* assistant pane is displayed in either of the following cases:

- Selecting the *View – Search* option from the Virtuoso® Hierarchy Editor window.
- Clicking the *Show All* option from the *Show Results* drop-down menu.
- Clicking the Right Mouse Button on the entry in the Table View and selecting the *Search Hierarchy For* option from the context-sensitive menu.
- Clicking the Right Mouse Button on the entry in the *Instantiations* list box of the Explain form and selecting the *Search Hierarchy For* option from the context-sensitive menu.

As compared to the HED toolbar search, which shows only the top 10 matching results, the *Search* assistant pane shows all the matching results. On selecting the cell or instance in the *Search* assistant pane, the corresponding entry is highlighted in the *Tree View* or *Table View* of the HED window.

Virtuoso Hierarchy Editor User Guide

Virtuoso Hierarchy Editor Overview

The total number of search hits are displayed at the top-left corner of the *Search* assistant pane. Placing the pointer over this number also displays the potential hit targets and how long it took to perform the current search. For example, “*49 hits from 727 targets, found in 0 seconds*”.

To view the demonstration on the search functionality, see [Using the Search Functionality in Hierarchy Editor](#) video.

Note: Access to this video depends on the availability of a web browser and a Cadence Online Support account.

Related Topics

[Starting the Hierarchy Editor](#)

[Hierarchy Editor Menu Commands](#)

[Hierarchy Editor Toolbar](#)

Using the Top Cell and Global Bindings Section in Hierarchy Editor

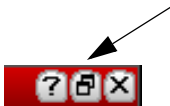
The Top Cell section is a dockable window. You can dock it on any side of the main Hierarchy Editor window or undock it to be a floating window.

To change the location of the Top Cell window,

- Drag it by the title bar and place it along any inside edge of the main window.

To float the Top Cell window,

- Click the *Float* button on the title bar of the window or double-click the title bar.



You can dock it by clicking the *Float* button again or by double-clicking the title bar.

To display or hide the Top Cell window,

- Select *View – Top Cell*.

The *Global Bindings* section contains the global library, view, and stop lists. These are the default values for the entire configuration.

Selecting the browse (...) button for a specific *Global Bindings* list option displays a view list building form that allows you to build up a view list without the need to manually type in the libraries, views, and so on.

The Global Bindings section is a dockable window. You can dock it along any side of the main Hierarchy Editor window or undock it to be a floating window.

To change the location of the Global Bindings window,

- Drag it by the title bar and place it along any inside edge of the main window.

To float the Global Bindings window,

- Click the *Float* button on the title bar of the window or double-click the title bar.

You can dock it by clicking the button again or double-clicking the title bar.

To display or hide the Global Bindings window,

- Select *View – Global Bindings*.

Virtuoso Hierarchy Editor User Guide

Virtuoso Hierarchy Editor Overview

Related Topics

[Virtuoso Hierarchy Editor Editing Form](#)

[View List Building Forms](#)

Displaying the Table View in Hierarchy Editor

The table view displays all the cells of your design in a table format.

To display the table view,

1. Select the *Table View* tab or, from the menu bar, choose *View – Parts Table*.
2. To display instances of the selected cell, from the menu bar, choose *View – Instance Table*.

Related Topics

[Cell Bindings Table in the Table View Tab](#)

[Performing Multiple Selection Using Cell Bindings Table](#)

[Instance Bindings Table in the Table View Tab](#)

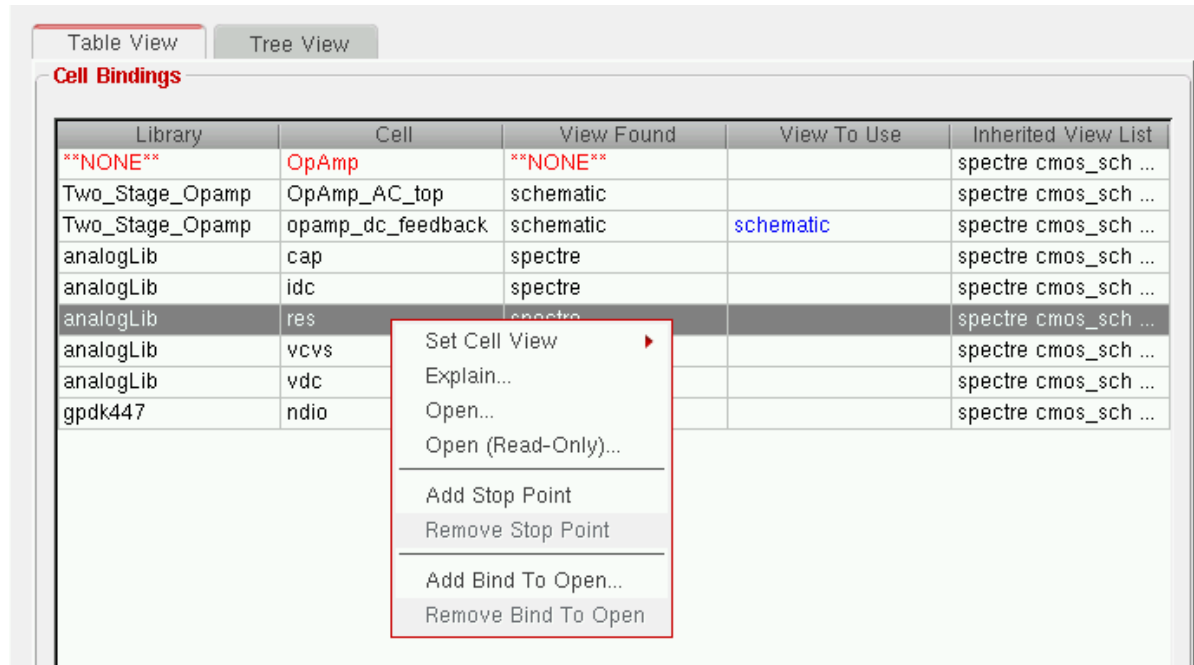
[Displaying the Tree View in Hierarchy Editor](#)

[Virtuoso Hierarchy Editor Editing Form](#)

Cell Bindings Table in the Table View Tab

The Cell Bindings table displays the library and view bindings of cells.

You can sort the cells in the Cell Bindings table by clicking on any of the column headings. You can also move the columns by clicking-and-dragging the column heading.



Mark As External HDL Text (AMS UNL only)

This option is used to bind a cellview to an external text view and is applicable only in AMS UNL flow.

This option has been added as another HED cellview binding property selection for use in the AMS UNL flow that can be used to bind a cellview to an external text file, which can be accessed or passed through `-v/-y/-f/-reflib` on the `xrun` argument.

The `irun` argument is also valid but will be removed in the future releases.

For more information on the AMS UNL flow, refer to [AMS Unified Netlister](#).

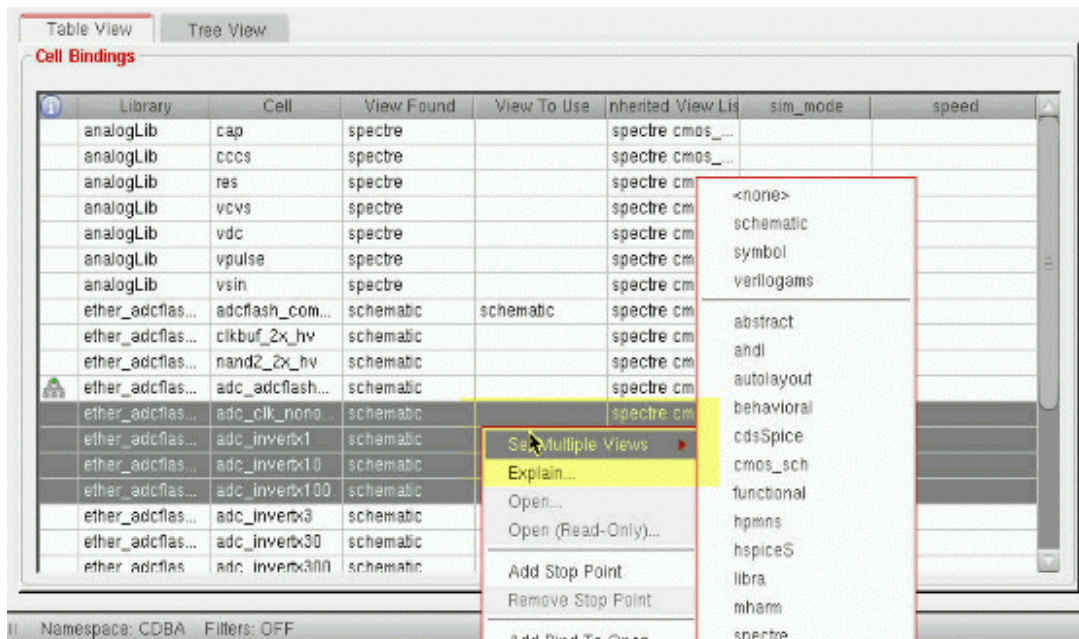
Cell Bindings Table Context-Sensitive Menu

Right-clicking over the contents of the *Cell Bindings* table displays a context menu that contains options specific for use with the cell bindings listed.

Multiple Selection and Action

You can select multiple items in the *Cell Bindings* table, and apply specific operations to all of them, such as setting a binding, applying property values, setting a view list, setting stop points, and so on, using the context-sensitive menu.

The Hierarchy Editor only applies the selected operation on those items where it makes sense to do so. For example, if an item is not editable it would be skipped.



Related Topics

[Performing Multiple Selection Using Cell Bindings Table](#)

[Instance Bindings Table in the Table View Tab](#)

[Displaying the Tree View in Hierarchy Editor](#)

[Virtuoso Hierarchy Editor Editing Form](#)

Performing Multiple Selection Using Cell Bindings Table

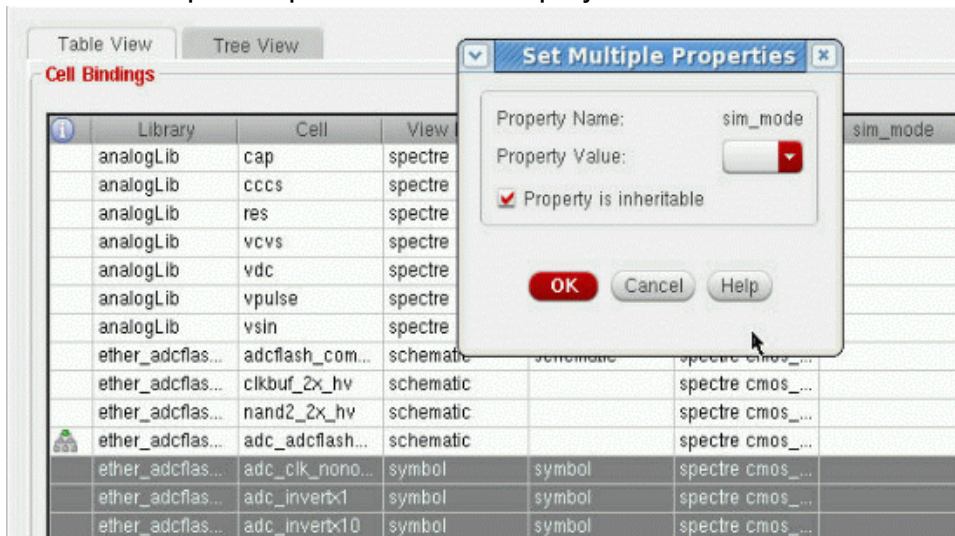
To perform a multiple selection and apply a chosen action:

1. Select the multiple items in the *Cell Bindings* table using **Shift** or **Ctrl** with the left-mouse button.
2. Right-click over the *View to Use* column to display the context-sensitive menu.
3. Select *Set Multiple Views*.
4. Select the required view to be used for the multiple items selected.

You can also apply the same technique to, for example, set a property value to a multiple selection of items:

1. Select *View – Properties* to display property information in the *Cell Bindings* table.
2. Select multiple items in the *Cell Bindings* table using **Shift** or **Ctrl** with the left-mouse button.
3. Right-click over the property column (in this example *sim_mode*) to display the appropriate context-sensitive menu.
4. Select *Set Multiple Properties*.

The Set Multiple Properties form is displayed.



5. Complete the Set Multiple Properties form as required.
6. Click **OK** to apply the changes to all selected items.

Virtuoso Hierarchy Editor User Guide

Virtuoso Hierarchy Editor Overview

Related Topics

[Cell Bindings Table in the Table View Tab](#)

[Instance Bindings Table in the Table View Tab](#)

[Displaying the Tree View in Hierarchy Editor](#)

[Virtuoso Hierarchy Editor Editing Form](#)

Instance Bindings Table in the Table View Tab

To display the *Instance Bindings* table, ensure that *View – Instance Table* has been selected, and that you also have an appropriate Lib/Cell/View selected in the *Cell Bindings* table.

Cell Bindings

Library	Cell	View Found	View To Use	Inherited View List
NONE	OpAmp	**NONE**		spectre cmos_s...
Two_Stage_Opamp	OpAmp_AC_top	schematic		spectre cmos_s...
Two_Stage_Opamp	opamp_dc_feedback	schematic	schematic	spectre cmos_s...
analogLib	cap	spectre		spectre cmos_s...
analogLib	idc	auCdl	auCdl	spectre cmos_s...
analogLib	res	spectre		spectre cmos_s...
analogLib	vcvs	spectre		spectre cmos_s...
analogLib	vdc	spectre		spectre cmos_s...
gpd447	ndio	spectre		spectre cmos_s...

Instance Bindings (Two_Stage_Opamp OpAmp_AC_top schematic)

Instance	Library	Cell	View Found	View To Use	Inherited View List
C0	analogLib	cap	spectre		spectre cmos_s...
I0	**NONE**	OpAmp	**NONE**		spectre cmos_s...
I2	analogLib	idc	auCdl	auCdl	spectre cmos_s...
I8	Two_Stage_Op...	opamp_dc_feed...	schematic	schematic	spectre cmos_s...
R0	analogLib	res	spectre		spectre cmos_s...
V1	analogLib	vdc	spectre		spectre cmos_s...
V2	analogLib	vdc	spectre		spectre cmos_s...
V4	analogLib	vdc	spectre		spectre cmos_s...
V44	analogLib	vdc	spectre		spectre cmos_s...

You can sort the instances in the *Instance Bindings* table by clicking on any of the column headings. You can also move the columns by clicking-and-dragging the column heading.

Right-clicking over the *Instance Bindings* table displays the same context-sensitive menu as displayed for the *Cell Bindings* table.

Related Topics

[Displaying the Table View in Hierarchy Editor](#)

[Cell Bindings Table in the Table View Tab](#)

Virtuoso Hierarchy Editor User Guide

Virtuoso Hierarchy Editor Overview

[Virtuoso Hierarchy Editor Editing Form](#)

[Virtuoso Hierarchy Editor Editing Form \(Table View\)](#)

[View List Building Forms](#)

Displaying the Tree View in Hierarchy Editor

The tree view displays the hierarchy of your design. It displays the top cellview and all the instances contained in it.

To display the tree view,

- ➔ Select the *Tree View* tab or, from the menu bar, choose *View – Tree*.

For more information about all the columns that can be displayed in the Cell Bindings table, see the description of the [Options Form \(Cell Table\)](#).

The tree view can be set to instance mode or occurrence mode. When in occurrence mode, any bindings, properties, and stop points that you specified for the occurrence. When in instance mode, these commands apply to the instance.

You can expand `verilog`, `verilogAMS`, `verilogA`, `systemVerilog`, `VHDL`, and `VHDLAMS` from the *Tree view* tab.

The default is instance mode; to set the occurrence mode, select the following icon on the toolbar:



Related Topics

[Virtuoso Hierarchy Editor Editing Form](#)

[Virtuoso Hierarchy Editor Editing Form \(Tree View\)](#)

Displaying the Status Bar and Window Border in Hierarchy Editor

The status bar, whose default location is at the bottom of the main window, displays the state of the Hierarchy Editor—the name space in which you are operating, whether filters are on or off, and whether an update is required.



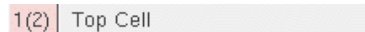
Namespace: CDBA Filters: OFF Update: Needed

You can move the status bar to any side of the main window. You cannot float it.

To move the status bar,

- ➡ Drag it by its handle and place it on any inside edge of the main window.

The window border displays the window number. For example, 1(2), where 1 is the session window number and 2 is the window number of the Hierarchy Editor.



1(2) | Top Cell

The window border also displays the command that is currently selected or that was last selected.

Related Topics

[Starting the Hierarchy Editor](#)

[Hierarchy Editor Toolbar](#)

[Using the Top Cell and Global Bindings Section in Hierarchy Editor](#)

[Displaying the Table View in Hierarchy Editor](#)

[Displaying the Tree View in Hierarchy Editor](#)

[Virtuoso Hierarchy Editor Editing Form](#)

Customizing Hierarchy Editor Menus

To add custom banner menus or context menu items to a Hierarchy Editor window:

1. Define a UI customization function that gets called whenever a new Hierarchy Editor window is opened and gets passed the `hiWindowId` of the new window.

The customization function can use `hi SKILL` functions to add banner menus to the Hierarchy Editor window or add menu items to the Hierarchy Editor context menus. The customization function can be loaded on Virtuoso startup through the `.cdsinit` file.

2. Register the customization function by using the `hedRegUICustomFunc SKILL` function. Alternatively, add this call to `.cdsinit` so that the customization function is registered during Virtuoso startup.

When customizing Hierarchy Editor context menus, the context menus can be accessed using the `cellTableMenu`, `instTableMenu`, and `treeMenu` properties for the Hierarchy Editor window.

The following properties are also available for the Hierarchy Editor window:

- ❑ `currentItem`: Returns the currently selected item in a Hierarchy Editor window.
- ❑ `libName`: Returns the library name of the current configuration.
- ❑ `cellName`: Returns the cell name of the current configuration.
- ❑ `viewName`: Returns the view name of the current configuration.
- ❑ `mode`: Returns the mode that the configuration was opened in. It can be either `r` (representing *read*) or `a` (representing *append*).

The following sample script defines customization functions that add a custom Hierarchy Editor banner menu and tree context menu item and registers the functions with Hierarchy Editor.

```
procedure (myBannerMenuUIFunc (hedWin)
  myMenu = hiCreateMenuItem(
    ?name 'myMenuItem
    ?itemText "My MenuItem"
    ?itemIcon nil
    ?callback "print (hiGetCurrentWindow()->currentItem) "
    ?disable nil
  )

  hiCreatePulldownMenu(
    'myMenu
    "My Custom menu"
    list(myMenu)
  )

  hiInsertBannerMenu(
```

Virtuoso Hierarchy Editor User Guide

Virtuoso Hierarchy Editor Overview

```
        hedWin
        'myMenu
        hiGetNumMenus(hedWin) + 1
    )
)

procedure(myContextMenuUIFunc(hedWin)
    let((treeMI)
        treeMI = hiCreateMenuItem(
            ?name 'treeMI
            ?itemText "My Tree MenuItem"
            ?itemIcon nil
            ?callback "print(hiGetCurrentWindow()->currentItem) "
            ?disable nil
        )
        hiAddMenuItem(hedWin->treeMenu treeMI)
    )
)

hedRegUICustomFunc('myBannerMenuUIFunc)
hedRegUICustomFunc('myContextMenuUIFunc)
```

Related Topics

[hedRegUICustomFunc](#)

[Customizing Hierarchy Editor Columns](#)

Customizing Hierarchy Editor Columns

The Hierarchy Editor lets you specify the columns you want to display, change the column order, and resize the columns.

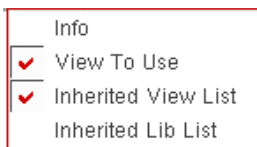
Specifying the Columns to Display

You can select the columns you want to display from the Options form or from a pop-up menu.

To select columns from the pop-up menu,

1. Right-click the column heading.

For example, the following pop-up menu appears when right-clicking over the *Cell Bindings* table.



The pop-up menu lists all the columns that can be displayed. Property columns appear in the list only if the *View – Properties* option is selected. Columns that are currently displayed have a check-mark next to them.

2. Select the columns that you want to display and deselect the ones that you want to hide.

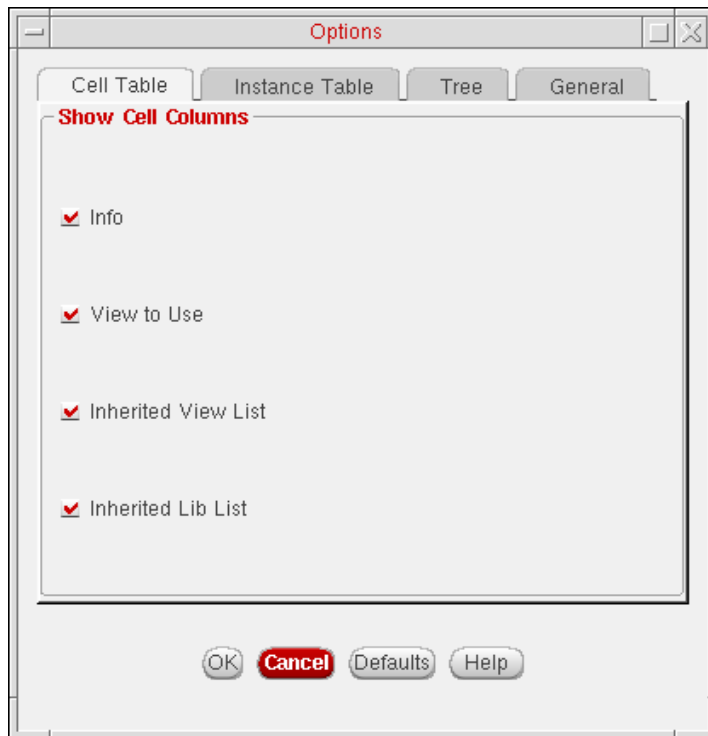
The tree view or table view display is updated to reflect your choices. These settings remain in effect for the session only unless you save your defaults.

The display/hide setting of individual property columns is not saved in the defaults file.

To specify columns in the Options form,

1. From the menu bar, choose *View – Options*.

The Options Form appears.



2. Click a tab to display a specific set of related attributes.

For example, to select the attributes you want to display in the tree structure, click the *Tree* tab.

3. Select the attributes you want to display. To select the default attributes, click *Defaults*.
4. Click *OK* to apply your changes and close the form.

Checking for unused configurations

You can check for unused configuration rules using the Checks tab in the Options Form. These rules can be HDB configuration rules such as a binding, stop point, or inherited view list that is set on some cell, instance, or occurrence, which is not a part of the current elaborated hierarchy.

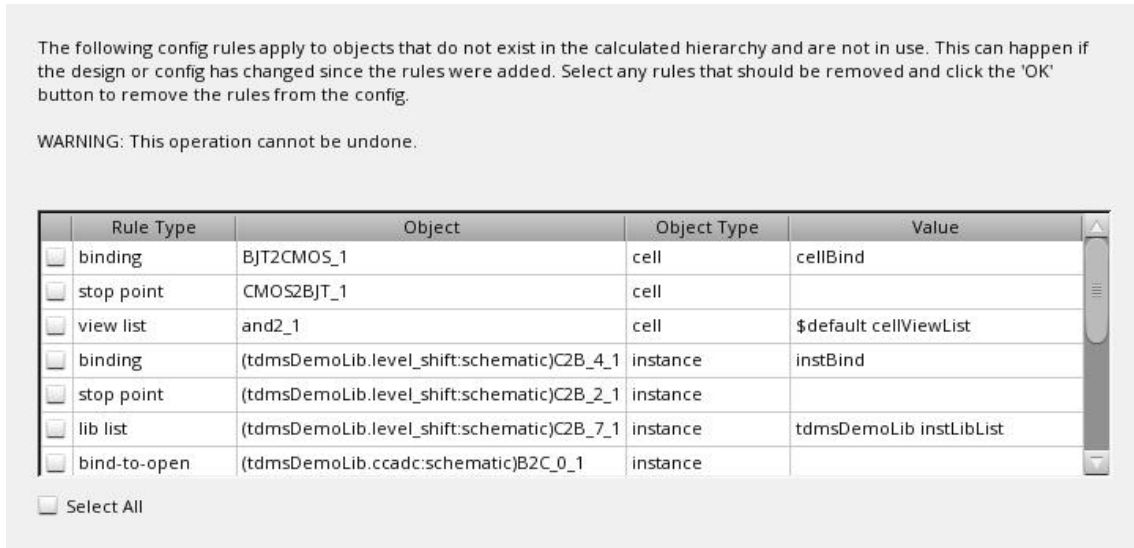
To perform check for unused configurations:

1. Open *Options – Check* after opening a configuration in edit mode.
2. In the Editing a configuration section, select when the check should be performed – *Open*, *Update*, or *Save*.

Virtuoso Hierarchy Editor User Guide

Virtuoso Hierarchy Editor Overview

The check runs for the selected operations and if unused rules are found then a dialog box appears with the list of unused rules.



3. Review and select the rules you want to delete from the configuration.
4. Click *OK*.

Changing Column Order

You can change the order of the columns in the table view or tree view. For example, you can have the *Cell* column appear before the *Library* column.

To move a column,

- ➡ Drag and drop the column heading to the new location.

Resizing Columns

You can change the width of any column by dragging the column heading.

To resize a column,

1. Place the cursor on the right border of the column heading till the double arrows appear.
2. Drag the border to the right to make the column wider or drag it to the left to make it narrower.

When you resize a column, the right-most column is adjusted to accommodate the new size. The other columns are not resized.

Sorting Data

You can sort data in the cell or instance table by column.

To sort by column,

1. Click the heading of the column by which you want to sort data. For example, to sort cells in the cell table by the views found, click the *View Found* column heading.

The column is sorted in alphabetical order.

2. Click the heading again to change the alphabetical order from ascending to descending or vice versa.

Related topics

[Options Form \(Cell Table\)](#)

[Options Form \(Instance Table\)](#)

[Options Form \(Tree\)](#)

[Options Form \(General\)](#)

[Options Form \(Checks\)](#)

Updating the Configuration Automatically

You can choose to have the configuration automatically updated whenever you make a change. For example, if you open a configuration and add a view to the global view list, the Hierarchy Editor automatically recomputes the hierarchy and display the updated bindings. If you do not set the automatic update option, you would need to click the *Update* icon or choose *View – Update* to view the changes you make.

The automatic update option is turned off by default.

To set the automatic update option,

1. From the menu bar, choose *View – Options*.

The Options form appears.

2. Select the *General* tab.
3. Select *Auto Update*.
4. Click *OK* to apply your change and close the Options form.

Related topics

[Options Form \(Cell Table\)](#)

[Options Form \(Instance Table\)](#)

[Options Form \(Tree\)](#)

[Options Form \(General\)](#)

[Options Form \(Checks\)](#)

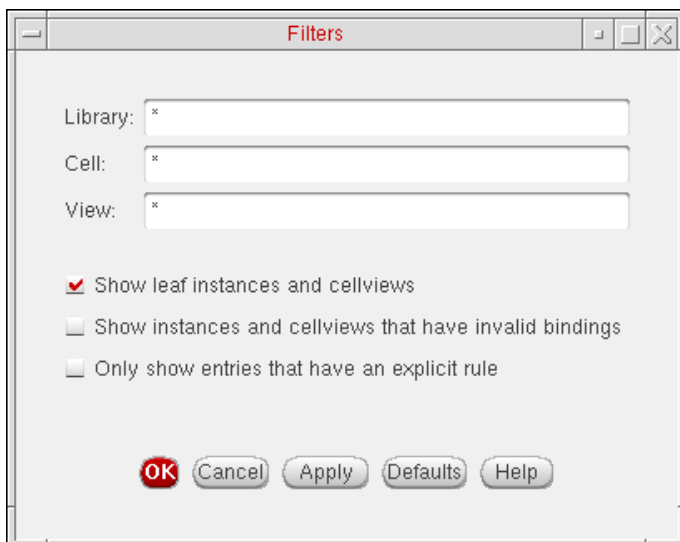
Filtering Cellviews

You can find cellviews quickly by using filters. You can set filters to display only those cellviews that match a certain pattern or those that belong to a particular library, for example. You can also choose to filter out cellviews that are at the leaf level, have an invalid binding, or do not have an explicit rule.

To filter cellviews,

1. Choose *View – Filters*.

The Filters form appears.



2. In the *Library*, *Cell*, and *View* fields, specify whole words or character strings to narrow your cellview search.

You can use the following wildcards:

Character	Match Criteria
?	Matches any single character
[<i>list</i>]	Matches any single character in <i>list</i>
[<i>lower-upper</i>]	Matches any character in the range between <i>lower</i> and <i>upper</i>
*	Matches any pattern

For example, to find all the libraries that begin with `p`, in the *Library* field, type `p*`.

Pattern matching is case sensitive. You can also specify multiple patterns by separating the patterns with spaces.

3. Select the type of data you want to see by selecting one or more of the following options:

- ☐ *Show leaf instances and cellviews*

Displays even those instances and cellviews that are at the leaf level of the tree, that is, they do not have a hierarchy under them.

- ☐ *Show instances and cellviews that have invalid bindings*

Displays even those instances and cellviews that are unbound or whose binding is invalid.

- ☐ *Only show entries that have an explicit rule*

Displays only the entries for which a view or view list has been set explicitly in the *View to Use* or *Inherited View List* column respectively, that is, the inherited value has been overridden. Entries with an explicit Bind-to-Open, Source File, or Reference Verilog setting are also displayed when this option is selected.

4. Click *Apply*.

The status bar displays *Filters: ON*.

5. Click *OK*.

The Filters form closes and the Hierarchy Editor uses the selected filter criteria until you turn off the filters.

The filter setting returns to the default setting when you start another Hierarchy Editor session. You must save your settings using the *File – Save Defaults* command to save the filter settings for future sessions.

Related topics

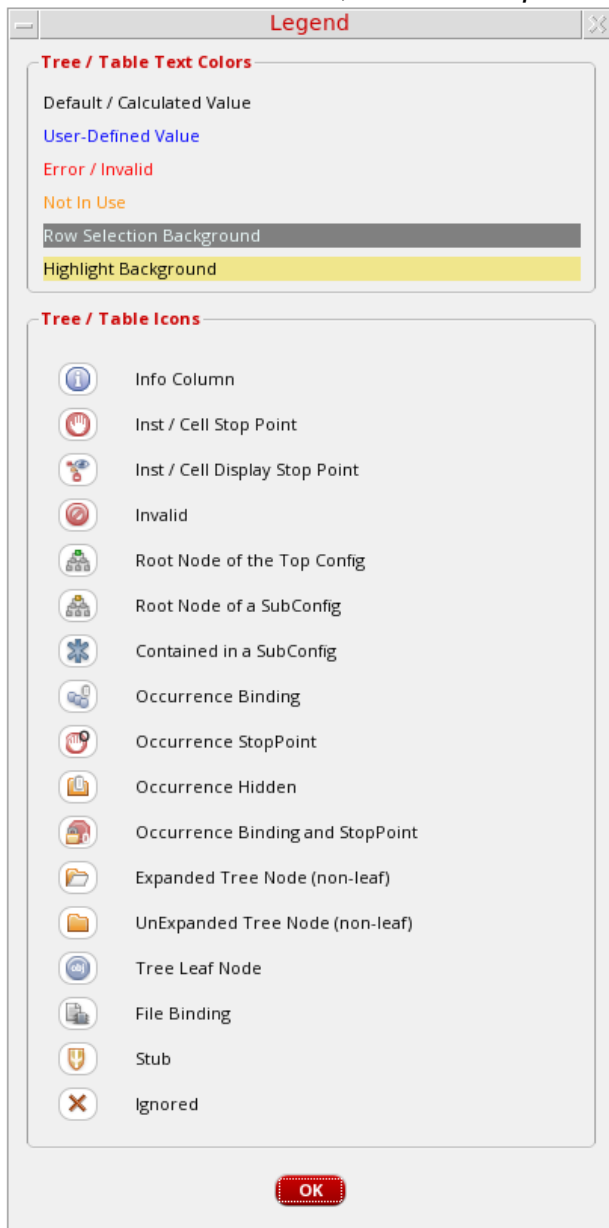
[Filters Form](#)

Viewing the Legend

The Legend dialog box provides information about the icons and colors that are used in the Hierarchy Editor user interface.

To display the Legend dialog box,

- ➔ From the menu bar, choose *Help – Legend*.



Virtuoso Hierarchy Editor User Guide

Virtuoso Hierarchy Editor Overview

Related topics

[Changing Binding Data Color Definitions](#)

Changing the Hierarchy Editor Fonts

To change the Hierarchy Editor fonts,

1. Create a file named `.hedinit` (for standalone HED) or `.cdsinit` (for Virtuoso HED) in your home directory or current working directory.
2. In the `.hedinit` or `.cdsinit` file, add an `hiSetFont` SKILL function call specifying the font to use.

The Hierarchy Editor only supports the `label` font type for the `t_fontType` argument.

For example:

```
hiSetFont("text" ?name "Roboto Mono" ?size 12)
hiSetFont("ciw" ?name "Roboto Mono" ?size 12)
hiSetFont("label" ?name "Open Sans" ?size 12)
```

3. Restart the Hierarchy Editor.

You can have multiple `.hedinit` or `.cdsinit` files. The Hierarchy Editor looks for the following files, in this order, and uses the first file that is found:

- `your_install_dir/tools/dfII/local/.hedinit`

Use this directory if you want to customize fonts for your entire site.

If you create a site `.hedinit` or `.cdsinit` file, you should include the following in the file:

```
if( and( isFile( "../.hedinit" ) isReadable( "../.hedinit" ) )
    then loadi( "../.hedinit" )
    else when( and( isFile( "~/hedinit" ) isReadable( "~/hedinit" ) )
        loadi( "~/hedinit" )
    )
)
```

This loads users' `../.hedinit` or if it exists, `~/hedinit` if that exists, and is needed because the Hierarchy Editor only reads the first `.hedinit` file that is found.

□ `../.hedinit`

□ `~/hedinit`

- `your_install_dir/tools/dfII/local/.cdsinit`

```
if( and( isFile( "../cdsinit" ) isReadable( "../cdsinit" ) )
    then loadi( "../cdsinit" )
    else when( and( isFile( "~/cdsinit" ) isReadable( "~/cdsinit" ) )
        loadi( "~/cdsinit" )
    )
)
```

Virtuoso Hierarchy Editor User Guide

Virtuoso Hierarchy Editor Overview

This loads users' `./cdsinit` if it exists, or `~/cdsinit`, if that exists, and is needed because the Hierarchy Editor only reads the first `.cdsinit` file that is found.

- ❑ `./cdsinit`
- ❑ `~/cdsinit`

Related topics

[Hierarchy Editor Menu Commands](#)

[Customizing Hierarchy Editor Menus](#)

[Customizing Hierarchy Editor Columns](#)

Exiting the Hierarchy Editor

To close the Hierarchy Editor,

1. Choose *File – Exit* or press `Control-q`.

The Hierarchy Editor closes.

If you have not saved the configuration changes you have made, a *Warning* form asks you whether you want to save your changes.

2. On the *Warning* form, click one of the following:

- ☐ *Yes* to save your changes and exit
- ☐ *No* to discard your changes and exit
- ☐ *Cancel* to dismiss the form without exiting

The Exit command closes only the Hierarchy Editor. It does not close other applications with which the Hierarchy Editor might have been communicating.

Saving Custom Settings

When you exit the Hierarchy Editor, your settings are not automatically saved. You can save your settings with the Save Defaults form.

You can save and restore the following settings:

- The height and width of the Hierarchy Editor window
- The location of the Hierarchy Editor window on the screen
- The filter settings
- The view options to show or hide the Top Cell window, Global Bindings window, the tree view or table view, Instance Bindings table, and properties
- The size of the instance table, tree structure, and message area
- The columns that are displayed, including property columns
- The Hierarchy Editor form values
- Plug-in settings

To save your current settings,

Virtuoso Hierarchy Editor User Guide

Virtuoso Hierarchy Editor Overview

1. Choose *File – Save Defaults*. The Save Defaults form appears.
2. Type the path to the directory in which you want to save the settings.
3. Click *OK*.

The information is saved in the `hed.env` file in the directory you specified.

The next time you start the Hierarchy Editor, it loads the `hed.env` file. If the `hed.env` file is not in a directory that is found by the Cadence search mechanism, you can load the file by starting the Hierarchy Editor with the `-restore dirname` command.

Related topics

[Starting the Hierarchy Editor](#)

[Filtering Cellviews](#)

[Customizing Hierarchy Editor Columns](#)

[Customizing Hierarchy Editor Menus](#)

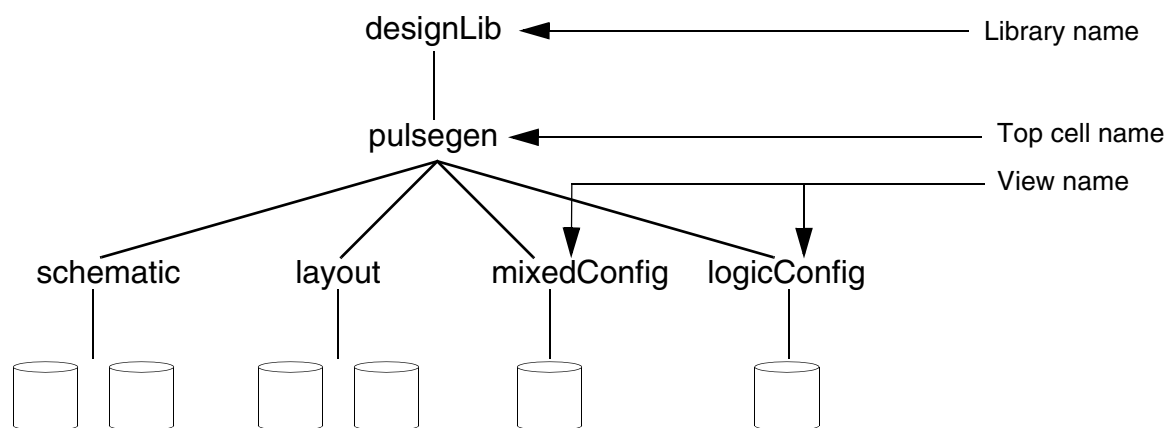
Design Hierarchy Configuration View

The configuration view of a design is the definition that the Hierarchy Editor understands and can open. It is a cellview that stores a configuration file that contains the libraries, cells, and views of a design.

You can browse:

- The library directory, which contains a collection of cells that correspond to a specific process technology
- The cell directory, which contains the design object that forms an individual building block of a chip or system
- The view, which is a defined representation of a cell such as layout or a schematic

The main components of a configuration are the top cellview, which is the root of the design and the configuration rules.



In the above example, the `designLib` library contains two configuration definitions identified by the view names `mixedConfig` and `logicConfig`. You can use the Hierarchy Editor to open the `mixedConfig` or the `logicConfig` configuration.

Virtuoso Hierarchy Editor User Guide

Design Hierarchy Configuration View

Related topics

[Opening Configurations in Hierarchy Editor](#)

Opening Configurations in Hierarchy Editor

To open an existing configuration from the command-line when you start the Hierarchy Editor,

- ➔ In a terminal window, type

```
cdsHierEditor -lib libname -cell cellname -view viewname
```

You must specify all of the above options for the Hierarchy Editor to open a configuration and load the data. If you do not specify a cellview (that is, the library, cell, and view names), the Hierarchy Editor opens without loading a configuration.

Note: If the Hierarchy Editor cannot open the configuration for editing—for example, if the configuration does not have edit permissions or if it is already locked by another process—it brings up a dialog box that asks if you want to open the configuration in read-only mode. Click *Yes* to open the configuration in read-only mode.

To open a configuration from the Hierarchy Editor,

1. From the menu bar, choose *File – Open* or press `Control-o`.

The Open form appears.

2. In the *Library* field, select or type the name of the library that contains the design.

The drop-down list contains all the libraries that are defined in your library definition file (`cds.lib`) file.

3. In the *Cell* field, select or type the name of the cell that contains the configuration view.

The drop-down list contains all the cells in the library you selected.

4. In the *View* field, select or type the name of the configuration view.

The drop-down list only displays configuration views of the cell that you selected; it does not display any other views.

5. Click *OK*.

The Hierarchy Editor opens the configuration.

You can use the `maskLayoutStopLimit` variable in the `hed.env` file to specify a size limit for maskLayout cellview databases. If a configuration includes such databases that are larger than the specified size limit, they do not get opened and gets shown as leaves with display stop point icons. These display stop points do not affect netlisting or simulation.

If the Hierarchy Editor cannot open the configuration for editing—for example, if the configuration does not have edit permissions or if it is already locked by another

Virtuoso Hierarchy Editor User Guide

Design Hierarchy Configuration View

process—it brings up a dialog box that asks if you want to open the configuration in read-only mode. Click *Yes* to open the configuration in read-only mode.

To open a configuration in read-only mode from the Hierarchy Editor,

1. From the menu bar, choose *File – Open (Read-Only)*.

The Open (Read-Only) form appears.

2. In the *Library* field, select or type the name of the library that contains the design.

The drop-down list contains all the libraries that are defined in your library definition file (`cds.lib`) file.

3. In the *Cell* field, select or type the name of the cell that contains the configuration view.

The drop-down list contains all the cells in the library you selected.

4. In the *View* field, select or type the name of the configuration view.

The drop-down list only displays configuration views of the cell that you selected; it does not display any other views.

5. Click *OK*.

The Hierarchy Editor opens the configuration.

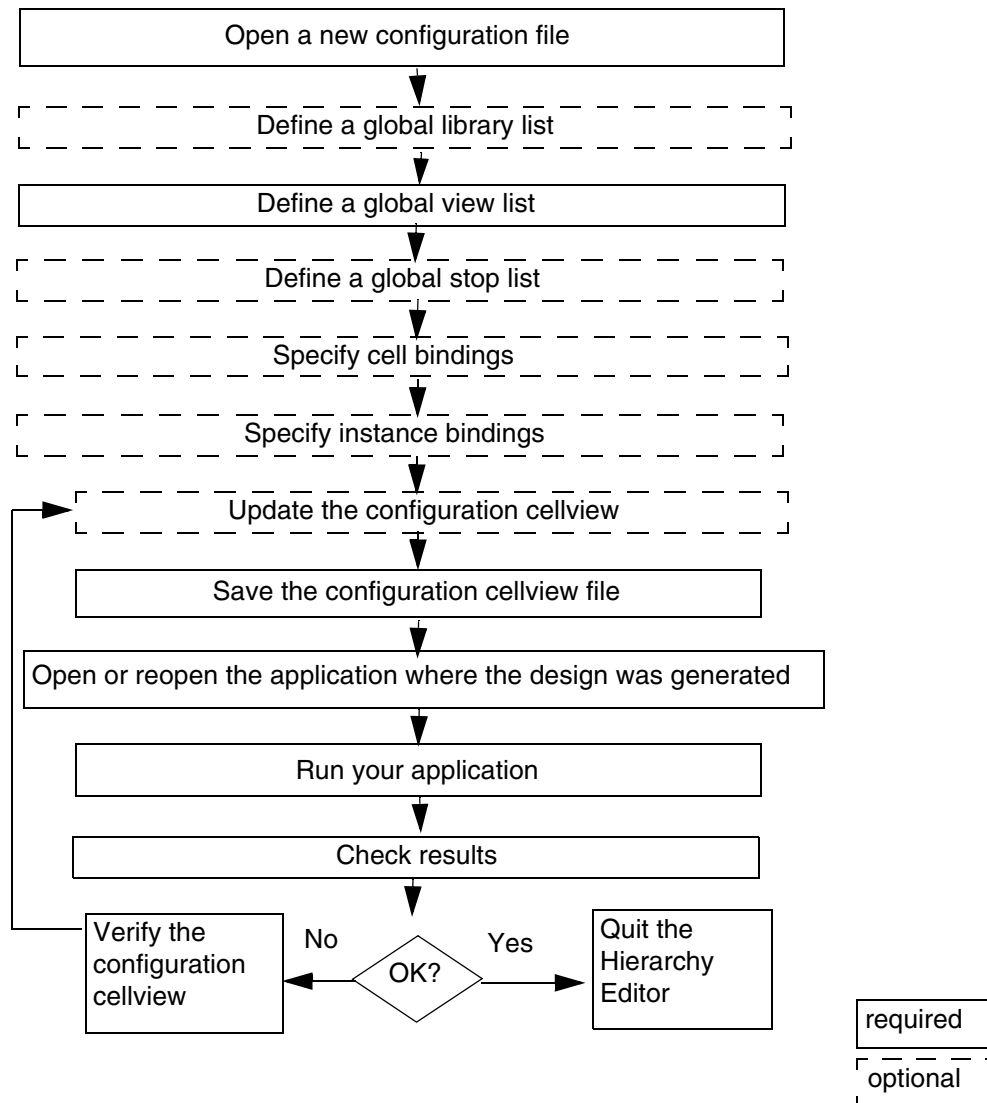
Related topics

[Open Form](#)

[Viewing the Description File](#)

Creating a New Configuration

Use these steps to create and use a configuration with the Hierarchy Editor:



To create a new configuration,

1. Choose *File – New Config* or press `Control-n`.

The New Configuration form appears.

2. In the *Top Cell* section, in the *Library*, *Cell*, and *View* fields, type or select the name of the library, cell, and view that you want to use as the top cellview of your design.
3. In the *Global Bindings* section, do one of the following:

Virtuoso Hierarchy Editor User Guide

Design Hierarchy Configuration View

- ❑ In the *Library List*, *View List*, *Stop List*, and *Constraint List* fields, specify the default bindings for the entire design.

The global library list is a list of libraries that determines the libraries from which each cell is obtained. The global view list is a list of views that determines which view is selected for each object in the design. The global stop list specifies a list of views that are to be treated as leaf nodes, that is, they are not to be expanded. The global constraint list is a list of constraint views that determines the constraints that apply to the design.

The *Stop List* and *Constraint List* fields are optional.

List the entries in each list in order of preference. Separate entries with spaces.

You can use constants in the view list and stop list. A constant is a symbolic name used to represent a set of views.

You can use the asterisk character (*) as a wildcard in the view list.

- ❑ Click *Use Template* to select a template that is compatible with the simulator you are running. Templates for simulators provide lists of views that are most often used for those simulators.

If the template you selected has filled in the *Top Cell* section of the New Configuration form, replace the names in the *Library*, *Cell*, and *View* fields with the top cellview you want to use.

4. (Optional) In the *Description* field, type a brief statement describing the new configuration.
5. Click *OK*.

The New Configuration form closes. The Hierarchy Editor displays the new configuration.

Related topics

[New Configuration Form](#)

[Using the Top Cell and Global Bindings Section in Hierarchy Editor](#)

[Creating and Editing Constants](#)

[Changing the Views in the View Choices List Box](#)

[Wildcards in a View List](#)

[Saving Configurations](#)

Editing Configurations in Hierarchy Editor

This section provides an overview of how to edit a configuration.

You cannot edit sub-configurations—a sub-configuration is a configuration that is contained within a configuration. To edit a sub-configuration, you must open it separately as a configuration in the Hierarchy Editor. For more information about sub-configurations, see [Hierarchy Editor Sub-Configurations](#).

To edit a configuration using the Hierarchy Editor,

1. Open the configuration.
2. Edit any of the following:
 - ☐ In the *Top Cell* section, you can change the top cellview of the configuration by editing the library name, cell name, or view name.
 - ☐ In the *Global Bindings* section, you can edit the library list, view list, stop list, and constraint list. Separate the entries in the lists with spaces.

The view list that applies to the root cellview of the design cannot be edited. In the *Cell Bindings* table, the root cellview is identified with a green pyramid in the *Info* column.

- ☐ Edit the Cell Bindings and Instance Bindings sections.
3. Choose *View – Update* to see the results of your changes in the *Cell Bindings* or *Instance Bindings* tables.

If you selected the *Automatic Update* option in the Options form, this step is not required because your configuration is automatically updated.

4. Click *File – Save* or press `Control-s`.

Other Cadence software tools cannot access new or changed configuration files until they have been saved.

Related topics

[Design Components in Hierarchy Editor](#)

[Virtuoso Hierarchy Editor Editing Form](#)

[Changing the Views in the View Choices List Box](#)

[Wildcards in a View List](#)

Editing the Cell Bindings and Instance Bindings Tables

To change cell or instance view bindings,

1. In the Cell Bindings or Instance Bindings table, select the cell or instance you want to change.
2. Do one of the following:
 - ☐ Click in the *View to Use* column of the cell or instance, type the new view, then press `Return`.
 - ☐ Right-click anywhere in the row of the cell or instance you want to change and select a view from the list of views in the pop-up menu.

The new view appears in the *View to Use* column in the color used to display user bindings.

3. Choose *View – Update* to see the results of your changes.

If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.

4. To save your changes, choose *File – Save* or press `Control-s`.

If the design or configuration changes, existing bindings or rules might no longer apply to the current hierarchy and stay unused. To detect and remove unused rules during various operations, choose *View – Options – Checks* and select the required options for the last check, which relates to unused configuration rules.

Adding Views to Cell or Instance Bindings

To add a new view,

1. In the Cell Bindings or Instance Bindings table, click in the *Inherited View List* column and move the cursor to the end of the list.
2. Type the name of the view you want to use. Separate entries with spaces.
3. Press `Return`.

The modified view list appears in the *Inherited View List* column in the color used to display user bindings.

4. To view your changes, choose *View – Update*.

If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.

5. To save the changes, choose *File – Save* or press `Control-s`.

Deleting Views from Cell or Instance Bindings

To remove any views from the cell or instance bindings,

1. In the *Cell Bindings* or *Instance Bindings* table, select the cell or instance you want to change.
2. In the *Inherited View List* column, double-click the view you want to remove.

The view is highlighted.

3. Press `Delete`.
4. To view your changes, choose *View – Update*.

If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.

5. To save the changes, choose *File – Save* or press `Control-s`.

Viewing Instances Contained by Other Instances

To see instances contained by other instances,

1. Select the *Tree View* tab to display the tree view of the Hierarchy Editor.
2. Right-click the instance you want to browse.

A pop-up menu appears.

3. Select *Expand Instance*.

The hierarchy below the instance is displayed.

Performing a Global Change on a Group of Instances

To perform a global view change on a group of instances, you can display all instances that share the same rules,

1. Choose *View – Options*.

The Options form appears.

2. Select the *Tree* tab.
3. In the *Expand Mode* section, select *By Instance Grouping*.
4. Click *OK*.
5. Choose *View – Tree* to display the tree view of the configuration.
6. Select the group of instances that you want to change.
7. Edit the *View to Use*, *Inherited View List*, or *Inherited Library List* column for the changes you want to make to the group of instances.
8. Press `Return`.

The change you made is applied to all the instances and appears in the color used to display user bindings.

9. To see the results of your changes, choose *View – Update*.

If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.

10. To save the changes, choose *File – Save* or press `Control-s`.

Viewing All Instances of a Subtree

To see all the instances of a subtree:

1. Select the *Tree View* tab.
2. Right-click the instance you want to browse and select *Expand Subtree* from the context-sensitive menu.

The hierarchy below the instance is displayed.

To close the hierarchy, right-click the instance and select *Collapse Subtree* from the context-sensitive menu.

Related topics

[Virtuoso Hierarchy Editor Editing Form](#)

[Changing the Views in the View Choices List Box](#)

Virtuoso Hierarchy Editor User Guide

Design Hierarchy Configuration View

Wildcards in a View List

Hierarchy Editor Sub-Configurations

A sub-configuration is a configuration that is used within another configuration. A sub-configuration is read-only within the configuration.

Cells contained in a sub-configuration are also displayed in the *Cell Bindings* table and are identified with an asterisk symbol in the *Information* column. The root cellview of the sub-configuration is identified with a brown pyramid in the *Information* column of the Cell Bindings table as well as in the Tree view.

You create a sub-configuration the way you create any other configuration.

You can then use the sub-configuration in your configuration in the same way that you use any other cellview.

After you have set a cell or instance to a sub-configuration, an update may need to be performed to see the latest contents of the sub-configuration.

To change the representation of a cell in your configuration to a sub-configuration

- ➔ Set the view of the cell or its instance to the configuration view by doing one of the following:
 - ☐ Specify the configuration view in the *View to Use* field of the cell or instance.
 - ☐ Right-click the cell or instance and use the *Set Cell View* or *Set Instance View* command on the pop-up to select the configuration view.

The cell or instance is then identified as a sub-configuration within your configuration.

A sub-configuration is read-only—you cannot edit it from the configuration it is contained in.

To edit a sub-configuration

- ➔ Open the sub-configuration in the Hierarchy Editor with the *File – Open* command and then edit it.

Related topics

[Creating a New Configuration](#)

Viewing the Description File

To view the description file,

- ➡ Choose *Edit – Description*.

The Edit Description form appears.

To edit the description of your configuration,

1. In the Edit Description form, edit the description in the *Description* field.
2. Click *OK*.

The Hierarchy Editor saves the edited description.

Related topics

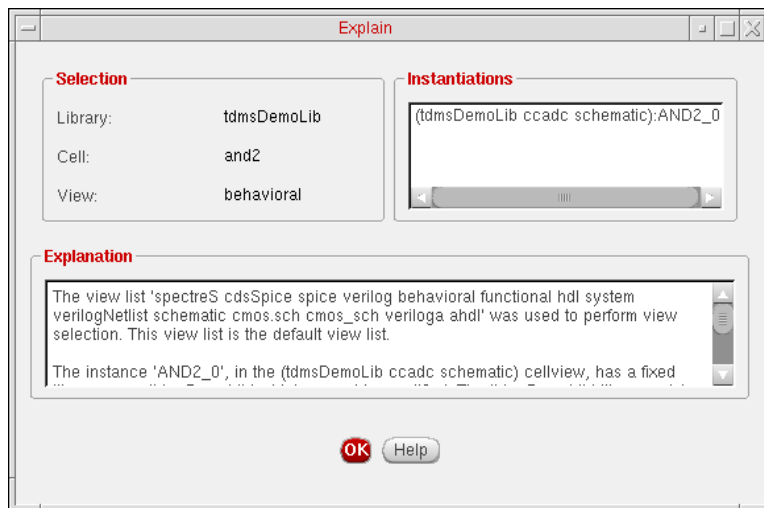
[Edit Description Form](#)

Verifying Binding Rules

To see which binding rules were used for a cell, do one of the following:

- In the *Cell Bindings* table, right-click a cellview and, from the pop-up menu, select *Explain*.
- In the *Cell Bindings* table, select a cellview and, on the toolbar, click the *Explain* icon.

The Explain form is displayed.



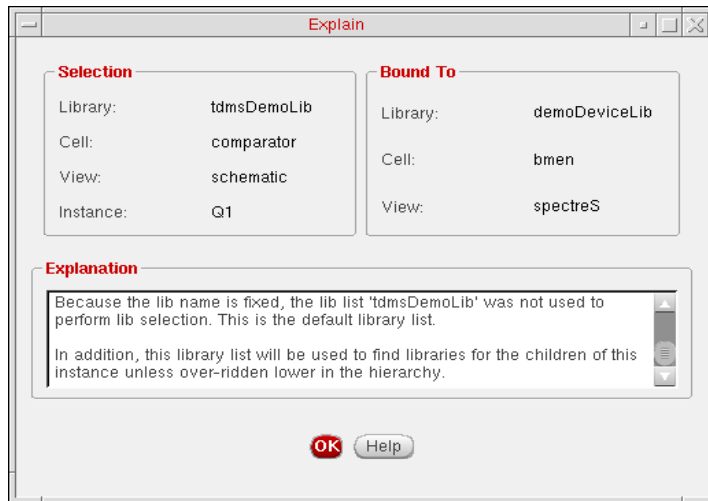
To see which binding rules were used for an instance, do one of the following:

- In the *Instance Bindings* table, right-click the instance whose rules you want to check and, from the pop-up menu, select *Explain*.
- In the *Instance Bindings* table, select the instance and, on the toolbar, click the *Explain* icon.

Virtuoso Hierarchy Editor User Guide

Design Hierarchy Configuration View

The Explain form appears.



To see which binding rules were used for an occurrence,

1. Click the *Tree View* tab to display the tree view of the configuration.
2. Right-click the occurrence whose rules you want to check.
3. From the pop-up menu, select *Explain*.

The Explain form appears.

Related topics

[Explain Form \(Selected Cell\)](#)

[Explain Form \(Selected Instance\)](#)

[Explain Form \(Selected Occurrence\)](#)

Working with Templates in Hierarchy Editor

Templates let you build configurations using pre-defined view lists, library lists, and stop lists. The Hierarchy Editor includes templates for some simulators that are compatible with Cadence software. You can also create your own templates.

Templates are located using CSF. The CSF search includes those templates in the installation hierarchy, and also any user-created templates that are found under `hierEditor/templates`, in directories that have been defined in `setup.loc`, for example `/hierEditor/templates/mytemplate`.

If any templates are found in one of these locations, they are displayed in the *Name* drop-down, without the need to specify a path.

If a template is not found in one of the locations mentioned, you can enter a path in the *From File* field.

Creating Templates

You can create your own template using your own view lists, library lists, stop lists, and constraints lists for your simulator or other design-specific requirements.

To create your own template,

1. In the Hierarchy Editor, choose *File – New Config* or press `Control-n`.

If you have made changes to the current configuration, the Hierarchy Editor prompts you to save your changes.

The New Configuration form appears.

2. In the *Top Cell* section, type the library, cell, and view name for the top cellview. These names are placeholders—you need to replace them when you use the template.
3. In the *Global Bindings* section, specify the library list, view list, stop list, and constraints list.

Separate entries with spaces.

4. (Optional) In the *Description* field, specify a brief description of the new template.
5. Click *OK*.

The New Configuration form closes.

The Hierarchy Editor displays the new configuration.

6. Choose *File – Save As*.

The Save As form appears.

7. In the Save As form, select or type the name of the library, cell, and view in which you want to save the new template.

The names do not have to match the names of the cell and view of the top-level cell.

8. Click *OK*.

The Save As form closes.

9. Open a terminal window.

10. Create a directory in which you want to place your templates.

11. Change directories to the `/library/cell/view` directory in which you saved the template.

12. Copy the configuration file in the directory (the `expand.cfg` file) to the directory you created. For example:

```
cp expand.cfg yourTemplatesDirectoryPath
```

When you use the new template, you need to specify the path to the template (in the *From File* field of the Use Template form). You can also copy your template to the location in which pre-defined Hierarchy Editor templates are stored.

Using Templates

To use a template,

1. Choose *File – New Config* or press `Control-n`.

The New Configuration form appears.

2. In the New Configuration form, click *Use Template*.

The Use Template form appears.

3. Do one of the following:

- ☐ If you want to use a pre-defined template, or a template that you have created and placed in the `your_install_dir/share/cdssetup/hierEditor/templates` directory, select the template from the *Name* listbox.

Template options available, dependent upon what tools are installed, are: *AMS*, *auCdl*, *auLvs*, *hSpiceD*, *spectre*, *spectreVerilog*, *systemVerilog*, *verilog*, and *vhdlInteg*.

- ❑ If you want to use a template that is not in the `your_install_dir/share/cdssetup/hierEditor/templates` directory,
 - Select *<Other>* from the *Name* listbox.
 - In the *From File* field, type the path to the template.

4. Click *OK*.

The Use Template form closes. The New Configuration form is filled in with the library list, view list, stop list, and constraint list data from the template.

5. In the *Top Cell* section, specify a top cellview for the new configuration.

6. Click *OK*.

The Hierarchy Editor displays the new configuration.

Copying a New Template to the Predefined Templates Location

You can copy the templates you create to the directory that contains pre-defined Hierarchy Editor templates and access them from the *Use Template* button on the *New Configuration* form.

To copy your template,

1. Change directories to the `/library/cell/view` directory in which you saved the template.

2. Copy the configuration file to the templates directory with the following command:

```
cp expand.cfg your_install_dir/share/cdssetup/hierEditor/templates/  
yourtemplatename
```

3. Restart the Hierarchy Editor.

Related topics

[Use Template Form](#)

[Cadence Search File mechanism](#)

Saving Configurations

When you save a configuration in the Hierarchy Editor, it is saved in the `expand.cfg` file in a configuration view. The configuration view directory also contains an `expand.cfg%` file, which is the previously-saved version of the `expand.cfg` file.

You can also save configurations in VHDL and Verilog®.

For VHDL, while for Verilog, it is in a file that contains paths to Verilog files for `xmvlog` to process them with the `-file (-f)` option.

Other applications cannot access changed configuration files until they have been saved in the Hierarchy Editor.

Important

If you access the standalone version of the Hierarchy Editor, it contains a designated message area at the bottom of the window. This is not however required in the Virtuoso-integrated version, where the Hierarchy Editor utilizes the CIW message area.

Saving a Configuration

To save a configuration,

- ➔ Choose *File – Save* or press `Control-s`.

The Hierarchy Editor saves the configuration.

To save a configuration using the *File – Save As* command,

1. Choose *File – Save As*.

The Save As form appears.

2. In the *Library* field, type or select the library in which you want to save the configuration.
3. In the *Cell* field, type or select the cell in which you want to save the configuration.
4. In the *View* field, type the name of the new configuration view.

The name you give to the cell and view of the configuration do not have to match the name of the cell and view of the top-level cellview.

5. Click *OK*.

Saving a Configuration as VHDL

The Hierarchy Editor provides the *File – Save As VHDL* command to save a configuration in VHDL syntax so that it can be read by VHDL tools. The VHDL configuration is always saved in a view called `configuration`.

The Hierarchy Editor adds those cellviews that have a VHDL source file to the VHDL configuration. It currently considers `vhdl.vhd` and `vhdl.vams` files as VHDL source files.

The Hierarchy Editor also adds those cellviews that do not have VHDL source files to the configuration but comments them out and generates a warning for them. However, if these cellviews have Verilog source files, you can choose to add them to the VHDL configuration by selecting the *Check for Verilog* option in the Save As VHDL form. If you select this option, the Hierarchy Editor adds the cellviews that do not have VHDL source files but do have Verilog source files to the VHDL configuration without commenting them out and does not generate a warning for them.

If Verilog views are included in the VHDL configuration based on the above, the inner `for/end` statements for these views are not written out.

To save a configuration for VHDL applications,

1. Choose *File – Save As VHDL*.

The Save As VHDL form appears.

2. In the *Library* and *Cell* fields, specify the name of the library and cell in which you want to save the VHDL configuration.

The *View* field is automatically set to `configuration`.

3. Select the *Check for Verilog* option if you want those cellviews that do not have VHDL source files but do have Verilog source files to be added to the VHDL configuration.

If you do not select this option, the Hierarchy Editor adds all the cellviews that do not have VHDL source files to the VHDL configuration but comments them out and generates a warning for them.

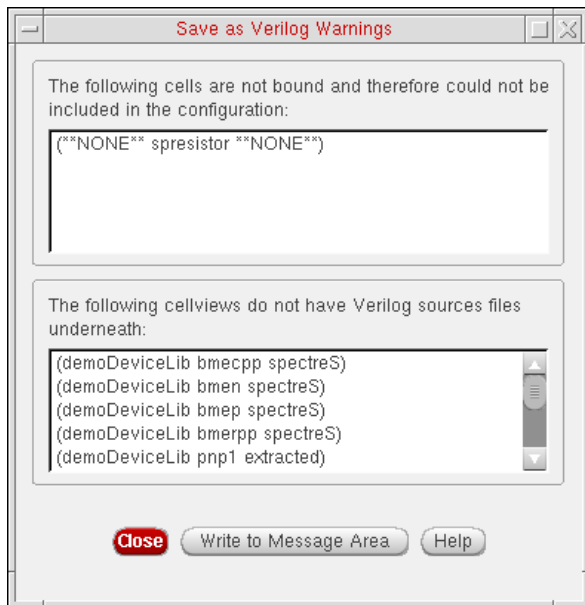
4. Click *OK*.

Saving a VHDL configuration does not save it in a format that can be read and edited by the Hierarchy Editor. You must also save the configuration with the *File – Save or File – Save As* command in order for it to be read by the Hierarchy Editor.

Virtuoso Hierarchy Editor User Guide

Design Hierarchy Configuration View

The VHDL configuration is saved. If the Hierarchy Editor generated any warnings while creating the VHDL configuration, it displays a dialog box asking you whether you want to see the warnings. If you click *Yes*, the Hierarchy Editor displays the following dialog box::



The Save as VHDL Warnings dialog box contains all the warnings that were generated while the VHDL configuration was created.

If you selected the *Check for Verilog* option when you saved the VHDL configuration, the warnings dialog box does not list those cellviews that do not have VHDL source files but do have Verilog source files.

To save the warnings to a log file,

- ➔ Click *Write to Message Area*.

The warnings are displayed in the Messages area and also saved to the log file.

Saving a Configuration as Verilog

The Hierarchy Editor provides the *File – Save As Verilog* command to save a configuration to a file that can be used with the `-file` argument of Verilog applications such as `xmvlog`. The Verilog file can have any name, though it is typically called `verilog.f`.

The Hierarchy Editor adds a list of Verilog source files to the Verilog file.

For any cellview, if the cellview is a Verilog view (as defined by the `master.tag` file), then the Verilog source file specified in the `master.tag` is used. (If the library has a `TMP` directory

Virtuoso Hierarchy Editor User Guide

Design Hierarchy Configuration View

associated with it, then the Hierarchy Editor looks for the source file in the temporary directory first, and then, if it is not found, looks for it in the master location.)

If the cellview is not a Verilog view (for example, if it is a schematic), then the Hierarchy Editor determines which Verilog source file to use. If the library has a `TMP` directory associated with it, the Hierarchy Editor first looks for source files in the temporary directory and then, if it does not find any, looks for them in the master location. If there are multiple source files, then the Hierarchy Editor selects the file according to the following order of precedence (highest to lowest):

```
verilog.vams  
verilog.va  
veriloga.va  
verilog.v
```

To save a configuration for Verilog applications,

1. Choose *File – Save As Verilog*.

The Save As Verilog form appears.

2. Specify the file in which you want to save the configuration:

- a. In the *Look in* field, select the directory in which you want to save the file.
- b. In the *File* field, type the name of the file.

The default value of this field is `verilog.f`.

3. Click *Save*.

The Verilog file is saved. If the Hierarchy Editor generated any warnings while creating the Verilog file, it displays a dialog box asking you whether you want to see the warnings.

To save the warnings to a log file,

- ➔ Click *Write to Message Area*.

The warnings are displayed in the Messages area and also saved in the log file.

Related topics

[Save As Form](#)

[Save As Verilog Form](#)

[Save As VHDL Form](#)

Virtuoso Hierarchy Editor User Guide

Design Hierarchy Configuration View

Using the Virtuoso Schematic Editor with the Hierarchy Editor

Comparing Configurations

You can compare two configurations and highlight their differences by opening the resulting design hierarchy side-by-side in a tree view.

To perform the comparison, you specify two configurations that are to be compared and traverse through their hierarchies. The traversal results are saved in the traversal output files, which are then compared to highlight differences between them.

Perform the following step to compare configurations:

1. Choose *File – Compare Configs*.

The *Compare Config* form appears.

2. Click *OK* to perform the comparison.

3. Click *Defaults* to populate the form fields with the default values from the `hed.env` registry file.

The comparison results are displayed in `tkdiff`. The output is displayed in a text format in a graphical tree structure and differences are highlighted. The traversal results for the same configuration can be different if you specify different values for the options. The options and their values are displayed in the traversal header as highlighted in the figure above.

Comparing Configurations from the Command-Line

You can also use the command-line interface to traverse and compare two configurations.

The command used to specify the comparison details is as follows:

```
hedConfigCompare [-outdir <dir>] [-s <numspaces>] [-min] [-subcfg]
[-difftool <toolcmd>] [-cdslib <path>] [-mlstoplimit <limit>]
<configSpec1> <configSpec2>
```

where,

- `-outdir <dir>`—Specifies a directory in which the traversal output files are saved. The default directory is `./configTraversals`.
- `-s <numspace>`—Specifies the number of spaces for indentation. The default value is 2 and the value ranges from 1 to 8.
- `-min`—Specifies the minimal traversal, which means traversing through hierarchy by avoiding revisits of the instances that have already been visited.

Virtuoso Hierarchy Editor User Guide

Design Hierarchy Configuration View

- `-subcfg`—Specifies whether the traversal results for subconfigurations are included in the output file.
- `-difftool <toolcmd>`—Specifies a tool to be used to display the compare traversal results. If this option is not specified, `tkdiff` is used to display the results.
- `-cdslib <path>`—Specifies the path from where `cdslib` file is to be loaded. If this file is not found, the command does not run.
- `-mlstoplimit <limit>` — Specifies the `maskLayoutStopLimit` value in MB, databases of `maskLayout` views exceeding the specified limit cannot be opened during the config comparison.
- `<configSpec1>` and `<configSpec2>`—These commands are described as follows:

```
{lib.cell:config [-output <outputPath>]} | {<inputPath>}
```

 - `lib.cell:config`—Specifies an elaborated and traversed configuration view.
 - `-output <outputPath>`—Specifies an output file in which the traversal results are saved. If the output file is not specified, the traversal results are saved in a file in the default output directory.
 - `inputPath`—If you do not want to perform the traversal for the configuration to be compared, you can directly provide the path to the output file.

Examples

- `hedConfigCompare lib1.cell1:config1 lib2.cell2:config2`

This command traverses both the configurations and save the traversal results in the following output files:

```
./configTraversals/lib1.cell1.config1
```

```
./configTraversals/lib2.cell2.config2
```

The command then compares these files.

- `hedConfigCompare -outdir ./localFiles lib1.cell1:config1 ./localFiles/config2.trav`

This command traverses only the first configuration and saves the results in default file in the specified output directory, `./localFiles`. The command then compares `./localFiles/lib1.cell1.config1` and `./localFiles/config2.trav`.

- `hedConfigCompare lib1.cell1:config1 -output ./localFiles/config1.trav ./localFiles/config2.trav`

Virtuoso Hierarchy Editor User Guide

Design Hierarchy Configuration View

This command traverses only the first configuration, `lib1.cell1:config1`, and saves the traversal results in the specified output file, `./localFiles/config1.trav` in the default output directory (`configTraversals`). This command then compares `./localFiles/config1.trav` and `./localFiles/config2.trav`.

- `hedConfigCompare lib1.cell1:config1 -output ./localFiles/config1.trav lib2.cell2:config2 -output ./localFiles/config2.trav`

This command traverses both the configurations and saves the results to the following files in the default output directory:

`./localFiles/config1.trav`

`./localFiles/config2.trav`

The command then compares these output files.

Related topics

[Compare Config Form](#)

Traversing Configurations

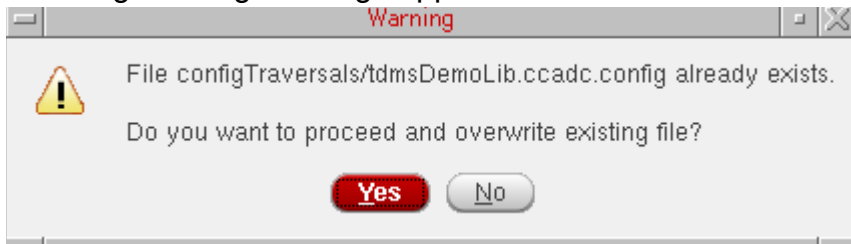
To traverse a configuration and save the results in an output file, do the following:

1. Choose *File – Traverse Config*.

The *Traverse Config* form appears.

2. Click *OK* to perform the traversal.
3. Click *Defaults* to populate the form fields with the default values from the `hed.env` registration file.

If you save the traversal results into an output file that already has the traversal output, the following warning message appears:



Traversing Configurations from the Command-Line

You can also use the command-line interface to traverse the configurations.

The command used to traverse through configuration is as follows:

```
hedConfigTraverse [-outdir <dir>] [-s <numspaces>] [-min] [-subcfg]
[-cdslib <path>] lib.cell:config [-output <outputPath>] [-mlstoplimit
<limit>]
```

where,

- `-outdir <dir>`—Specifies a directory in which the traversal output files are saved. The default directory is `./configTraversals`.
- `-s <numspace>`—Specifies the number of spaces for indentation. The default value is 2 and the value ranges from 1 to 8.
- `-min`—Specifies the minimal traversal, which means traversing through hierarchy by avoiding revisits of the instances that have already been visited.

Virtuoso Hierarchy Editor User Guide

Design Hierarchy Configuration View

- `-subcfg`—Specifies whether the traversal results for subconfigurations are included in the output file.
- `-cdslib <path>`—Specifies the path from where `cdslib` file is to be loaded. If this file is not found, the command does not run.
- `-mlstoplimit <limit>`— Specifies the `maskLayoutStopLimit` value in MB, databases of `maskLayout` views exceeding the specified limit cannot be opened during config traversal.
- `{lib.cell:config [-output <outputPath>]}`
 - `lib.cell:config`—Specifies an elaborated and traversed configuration view.
 - `-output <outputPath>`—Specifies an output file in which the traversal results are saved. If the output file is not specified, the traversal results are saved in a file in the default output directory.

Related topics

[Traverse config Form](#)

Using the Virtuoso Schematic Editor with the Hierarchy Editor

The Hierarchy Editor and the schematic editor can be kept synchronized when they are both started in the Virtuoso Studio design environment.

When you change the configuration in the Hierarchy Editor, the changes are not reflected in the schematic automatically. To update the schematic, you need to do the following:

1. In the Hierarchy Editor, save the configuration.
2. In the Hierarchy Editor, choose *View – Update* to update the configuration.

The configuration is re-read from disk and the schematic displays your changes.

If you do not save the configuration after making your changes, the following form appears when you choose *View – Update* in the Hierarchy Editor:



In the form,

1. Select the cellviews.
2. Click *OK*.

The configuration is saved and the schematic is also updated.

Virtuoso Hierarchy Editor User Guide

Design Hierarchy Configuration View

When you change the schematic, the changes are reflected in the Hierarchy Editor when you update the configuration.

When you start the Hierarchy Editor stand-alone and open a schematic from it, any changes you make to the schematic are not reflected in the Hierarchy Editor even when you update the hierarchy. For the Hierarchy Editor and the schematic editor to be synchronized, you must start them from the Virtuoso Studio design environment.

Related topics

[Starting the Hierarchy Editor](#)

Update Process for Read-Only Libraries in pc.db Files

The Hierarchy Editor (HED) follows the parent-child information of cellviews to traverse a design hierarchy. For text views, this information is found in their `pc.db` files.

To simplify this model, a new automatic mechanism has been introduced that does not require you to intervene in the `pc.db` file update process. The `pc.db` file is automatically created and saved to the `$CWD/.pcdb` directory when HED detects that it is either missing or out-of-date. HED finds and reads the `pc.db` file from this cache location. The generated `pc.db` files are also copied to the library area, if the library is writable.

It is not recommended, but you can change the location of `.pcdb` directory by setting the following environment variable:

```
setenv PCDB_CACHE_DIR <writable directory>
```

If you are currently using explicit TMP location in `cds.lib`, the following sequence is followed:

1. HED reads the `pc.db` files from the cache location.
2. If the `pc.db` files are not found in the cache location, HED reads the `pc.db` files from the explicit TMP location.
3. If the `pc.db` files are not found in the explicit TMP location, HED reads the `pc.db` files from the library area.

The explicit TMP is becoming obsolete and setting the TMP attribute in any `cds.lib` file is not recommended anymore for RO libraries.

If `pc.db` files are maintained as golden data in the RO libraries and explicit TMP is not being used in `cds.lib`, the new capability makes the whole flow—from configuration to netlist to simulation—to work automatically in the rare case when `pc.db` files become out-of-date.

The tool currently issues an error when it detects that a `pc.db` file is out-of-date and cannot update this `pc.db` in a RO directory.

Related topics

[Opening Configurations in Hierarchy Editor](#)

[Editing Configurations in Hierarchy Editor](#)


Creating a Snapshot Configuration

The snapshot configuration feature provides a way to freeze a certain configuration where all cells are explicitly bound in the whole design hierarchy. When this frozen configuration is reloaded, changes on a disk do not affect the binding when the library and view list are applied.

The snapshot configuration is served just like a generic configuration but with all the cells explicitly bound. It can be created, saved, edited, and loaded like a generic configuration. It is not supported in standalone HED.

Note: Before creating a snapshot configuration, you need to save changes (if any) to the original configuration.

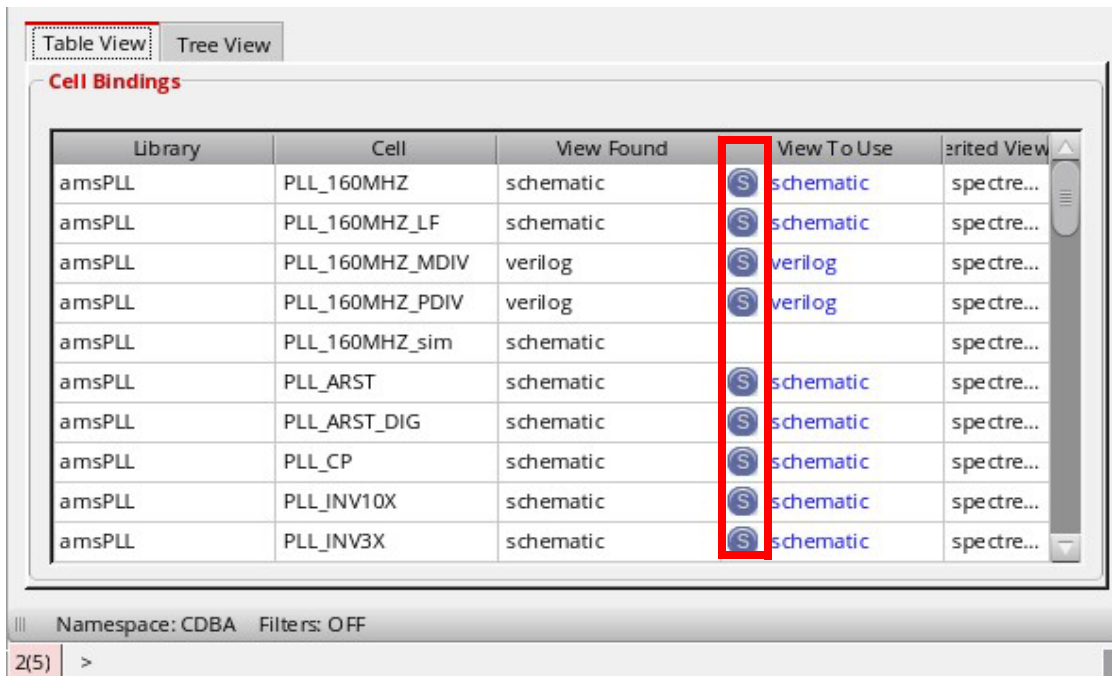
To create snapshot configuration, you need to perform the following steps:

1. From the Virtuoso Hierarchy Editor window, select *Edit – Create Snapshot Config*. Otherwise, you can select the *Create a snapshot config* icon () on the HED toolbar.
2. The creating a snapshot config dialog box is displayed. The default format of snapshot name is <config_name>_snapShot_<date>_<time>. In the *Description* section, the information is carried over from the original configuration. You can update the view name and description as required.

Virtuoso Hierarchy Editor User Guide

Design Hierarchy Configuration View

- Click the *OK* button, to create a fully bound snapshot config. As highlighted below, the snapshot binding are shown with the “S” symbol.



The cells or instances with “S” symbol imply the binding is set by the snapshot feature, whereas, those without “S” imply that it is explicitly set. The top design is "amsPLL" "PLL_160MHZ_sim" "schematic" and does not have an explicit binding set by the create snapshot functionality.

To get an accurate snapshot config for a design containing Pcells, open the Options form by selecting *View – Options* from the Virtuoso Hierarchy Editor window. Then, in the *General* tab, select the *Evaluate Pcells* check box, as highlighted below. If the Pcell evaluation is turned off, a warning message gets displayed when the snapshot is created or checked.

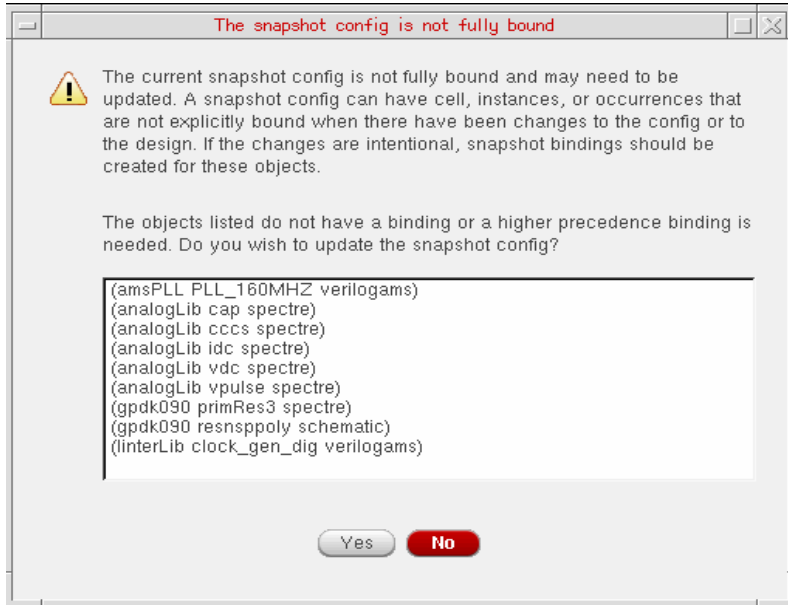
Checking a Snapshot Configuration

This option manually checks for the cells and instances that do not have an explicit binding. While updating the snapshot, if a cell is not explicitly bound with the right precedence level, then it gets updated with the same flow used to the set bindings when creating a snapshot.

Virtuoso Hierarchy Editor User Guide


Design Hierarchy Configuration View

For example, if the snapshot configuration is not fully bound then the following warning message is displayed.

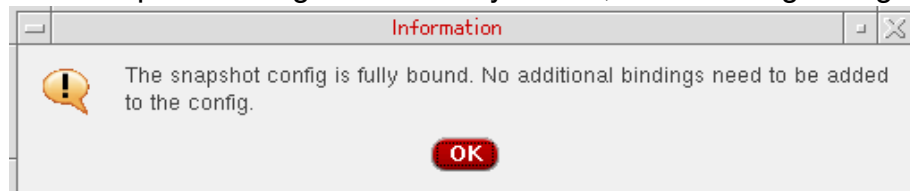


To add binding for the objects listed in the dialog box, click the *Yes* button.

Note: Any invalid cellviews or unused bindings should be corrected prior to updating a snapshot.

To check the snapshot config, select *Edit – Check Snapshot Config* from the Virtuoso Hierarchy Editor window. Otherwise, you can also select the *Check a snapshot config* icon () on the HED toolbar.

If the snapshot configuration is fully bound, the following dialog box is displayed.



HED does not create snapshot bindings for sub-configurations. Ideally, you should first create the snapshot sub-configurations and then use these snapshot sub-configurations in the top-level configuration. A warning message is displayed if non-snapshot sub-configurations are being used by a top-level snapshot configuration.

Clearing Snapshot Bindings

This option is used to remove all snapshot bindings. To clear the snapshot bindings:

Virtuoso Hierarchy Editor User Guide

Design Hierarchy Configuration View

1. Select *Edit – Clear Snapshot Binding* from the Virtuoso Hierarchy Editor window. The Confirm Clear Snap Shot dialog box is displayed.
2. Click *YES* to clear the snapshot bindings.

Related topics

[Virtuoso Hierarchy Editor Editing Form](#)

[Virtuoso Hierarchy Editor Editing Form \(Table View\)](#)

Virtuoso Hierarchy Editor User Guide

Design Hierarchy Configuration View

Design Components in Hierarchy Editor

With the Hierarchy Editor, you can control the expansion of your design for any given purpose such as simulation or netlisting. You do this by creating and editing rules that define a design configuration.

You can specify the following configuration rules in the Hierarchy Editor:

- **Library Binding:** Library binding determines the library from which a cell is obtained.
- **View Binding:** View binding determines which view of a cell is to be used in the configuration. For example, a cell might have different representations such as schematic, Verilog, and VHDL; the view binding determines which of these representations is used in the configuration.
- **Stop Lists and Stop Points:** Stop lists and stop points prevent further expansion of any part of a design.
- **Bind-to-Open:** The bind-to-open attribute is a way of specifying that a single instantiation of a cell is to be skipped by setting a bind-to-open attribute on it.
- **Constraint Binding:** Constraint binding determines which constraint view is to be used in the configuration.

You create and edit these rules at different levels of the design. You can specify global rules that apply to every level of the design. You can also specify rules at the cell, instance, and occurrence levels.

Note: Constraint binding can only be defined at the global level.

Global Bindings

Global bindings are configuration rules that are defined at the global level.

At the global level, you can define a library list, view list, and stop list. These global bindings become the default rules for the configuration and apply to every level of the hierarchy, unless they are overridden at lower levels of the hierarchy.

Cell Bindings

Cell bindings are configuration rules that are defined at the cell level. Cell bindings override global bindings for a cell.

Cell bindings apply to all instantiations of a cell.

You can define the following rules at the cell level:

- View binding that determines which view of the cell is used in the configuration

You can define view bindings for a cell in the following ways:

- ☐ Binding rules for a specific cell that apply to that cell and to components below it in the hierarchy, until they are overridden by another binding
- ☐ Binding for a specific cell that applies only to that cell and is not inherited by components below the cell
- Library binding that determines the library from which the cell is obtained. This is done by specifying a library list for the cell. Cell-level library lists are inherited by components below the cell in the hierarchy.
- Stop point on the cell that prevents the cell from being expanded.
- Bind-to-open attribute specifies that a single instantiation of a cell is to be skipped.

Instance Bindings

Instance bindings are configuration rules that are defined at the instance level. Instance bindings override global bindings and cell bindings for an instance.

Instance bindings apply to a single instantiation of a cell. Note, however, that if the cell that contains the instance is used in multiple places in the design, the binding applies to the instance in all those places. If you want to specify a binding for only one instance at a specific path, you need to specify occurrence bindings instead of instance bindings.

You can define the following rules at the instance level:

- View binding that determines which view of the cell is used in the configuration

You can define view binding for an instance in the following ways:

- ☐ Binding rules for a specific instance that apply to that instance and to components below it in the hierarchy, until they are overridden by another binding

- ❑ Binding for a specific instance that applies only to that instance and is not inherited by components below it in the hierarchy
- Library binding that determines the library from which the instance is obtained. This is done by specifying a library list for the instance. Instance-level library lists are inherited by components below the instance in the hierarchy.
- Stop point on the instance that prevents the instance from being expanded.
- Bind-to-open attribute that specifies that the instance is unbound, that is, it is not bound to a particular library, cell, or view.

Occurrence Bindings

Occurrence bindings are configuration rules that are defined at the occurrence level. An occurrence is an instance that is defined by the full path from the top-level design to the instance. Therefore, occurrence bindings apply to a single object at a specific path in the design.

You can specify the following rules at the occurrence level:

- Library, cell, and view binding for a single object at a specific path in the design
- Stop point that prevents the occurrence from being expanded
- Bind-to-open attribute that specifies that the occurrence is unbound, that is, it is not bound to a specific library, cell, or view.

Note: Not all Cadence tools support occurrences. Refer to the documentation for the applications you are using to check if those applications support occurrences.

Define a Library for an Object

You can define library binding rules at different levels of a design. The Hierarchy Editor uses the following order of precedence (listed from highest precedence to lowest precedence) to select the library for an object:

1. Occurrence binding
2. Instance-level Inherited Library List
3. Cell-level Inherited Library List
4. Global Library List

Define a View for an Object

You can define view binding rules in different ways and at different levels of a design. The Hierarchy Editor uses the following order of precedence (listed from highest precedence to lowest precedence) to select the view for an object:

1. Occurrence binding
2. Instance-level View to Use binding
3. Cell-level View to Use binding
4. Instance-level Inherited View List
5. Cell-level Inherited View List
6. Global View List

Related Topics

[Viewing the Description File](#)

[Rules Definition at the Global Level](#)

[Rules Definition at the Cell Level](#)

[Rules Definition at the Instance Level](#)

[Rules Definition at the Occurrence Level](#)

Rules Definition at the Global Level

You can define rules at the global level that become the default rules for the entire configuration. These rules are inherited by every level of the hierarchy until they are overridden by rules lower in the hierarchy.

You specify global rules, referred to as global bindings, in the *Global Bindings* section of the Hierarchy Editor. You can specify these rules in either the table view or the tree view of the Hierarchy Editor.

You can specify the following rules at the global level:

- Library List
- View List
- Stop List
- Constraint List

Related Topics

[Defining a View List at the Global Level](#)

[Defining a Library List at the Global Level](#)

[Defining a Stop List at the Global Level](#)

[Defining a Constraint List at the Global Level](#)

[Rules Definition at the Cell Level](#)

Defining a View List at the Global Level

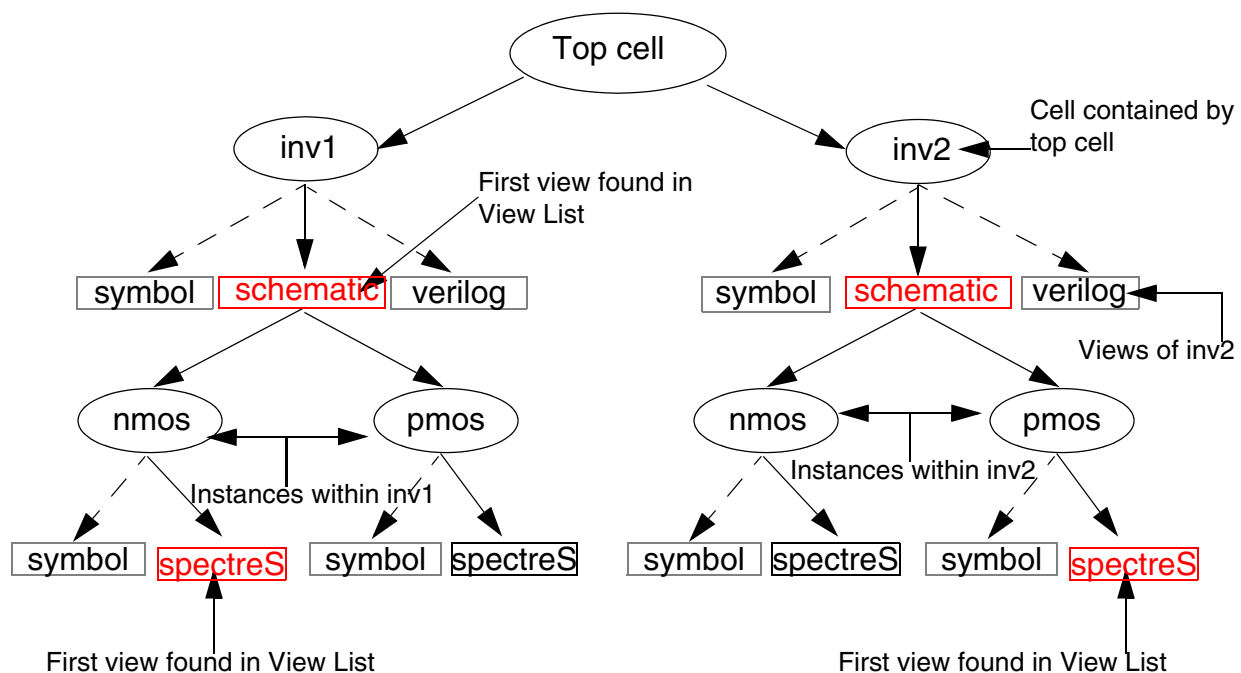
When you define views at the global level, the views apply to every level of the configuration and must be listed in the order in which you want them searched.

For example, if your view list is

```
spectreS schematic cmos.sch verilog
```

the Hierarchy Editor searches for each instance as shown in the figure.

View List: spectreS schematic cmos.sch verilog



The figure shows how the Hierarchy Editor searches for the first view listed in the View List. For example, in the cell `inv1`, the `symbol`, `schematic`, and `verilog` views are in the table. If the Hierarchy Editor finds a match, it selects that view for the design. If it does not find a match, it continues to look until a match is found. In the example, the second view in the view list, `schematic`, is the first view that is found.

The view list determines which view is selected for every object in the design, unless overridden by a cell binding, instance binding, or occurrence binding.

If the Hierarchy Editor does not find a view for a cell or instance, it displays ****NONE**** for that cell or instance in the *View Found* column of the Cell or Instance Bindings table.

****NONE**** indicates a binding error that prevents you from netlisting until you correct the error.

The global view list that you define becomes the default Inherited view list for each cell and instance. You can override the default at the cell, instance, or occurrence levels. You can also check which binding rules were used to select the view for an object. For more information, see [Viewing the Description File](#).

Creating a Global View List

To create or change the global view list,

1. In the *Global Bindings* section, click the *View List* field.
2. Type the views you want in the order of preference. Separate the entries with a space.

You can also define a constant, such as `$default`, `$digital`, or `$analog`, to represent a list of views and use this constant in the view list. Constants are easier to read than a long view list, and they can be re-used in any view list. You can also use the asterisk character (*) as a wildcard in the view list.

Specify the view name of the Top cell when defining `$default` to generate a complete view list including the view of the Top cell.

3. Press `Return`.
4. To see the changes in the configuration, click the *Update* icon in the toolbar.

The *Tree View* and *Table View* change to show the new configuration resulting from the change in the View List.

If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.

5. To save the changes, from the menu bar, choose *File – Save* or press `Control-s`.

Deleting Views from the Global View List

To delete a view from the global view list,

1. In the *Global Bindings* section, click in the *View List* field.
2. Highlight the view you no longer want and press `Delete`.
3. To see the changes in the configuration, click the *Update* icon in the toolbar.

Related Topics

[View List Building Forms](#)

[Changing the Views in the View Choices List Box](#)

[Wildcards in a View List](#)

[Library Scoped Views in a View List](#)

Defining a Library List at the Global Level

Library lists determine the library (assuming that the library is not fixed by, for example Virtuoso) from which each component in the design is obtained. Libraries are listed in order of preference. The Hierarchy Editor searches the libraries in the order in which they are listed and uses the component from the first library in which it is found.

Global library lists apply to every level of the configuration. Global library lists can be overridden by library lists at the cell and instance levels and by specific library cell:view binding on an occurrence.

You can see which binding rules were used to select a library for a component. For more information, see [Viewing the Description File](#).

Creating a Library List at the Global Level

To create or change a library list at the global level,

1. In the *Global Bindings* section of the Hierarchy Editor, click in the *Library List* field.

2. Do one of the following:

- ☐ To create a new library list, type the libraries in the order of preference or use one of the View List Building Forms.
- ☐ To add a library to an existing library list, type the new library in the appropriate place. The libraries are searched in the order in which they are listed.
- ☐ To remove a library from the library list, highlight the library and press `Delete`.

3. Press `Return`.

4. To see the changes in the configuration, click the *Update* icon in the toolbar.

If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.

5. To save the changes, choose *File – Save* or press `Control-s`.

Related Topics

[View List Building Forms](#)

[Rules Definition at the Global Level](#)

Virtuoso Hierarchy Editor User Guide

Design Components in Hierarchy Editor

[Defining a View List at the Global Level](#)

[Defining a Stop List at the Global Level](#)

[Defining a Constraint List at the Global Level](#)

Defining a Stop List at the Global Level

A stop list is a list of views that are to be treated as if they are at the leaf level, that is they are to be treated as if they do not have any levels of hierarchy below them and cannot be expanded further. The stop list is optional—the Hierarchy Editor does not require you to have a stop list in your configuration.

You can use the stop list to designate cellviews that contain no instances. In addition, you can use the stop list to indicate cases where you do not want the configuration to descend further into the hierarchy.

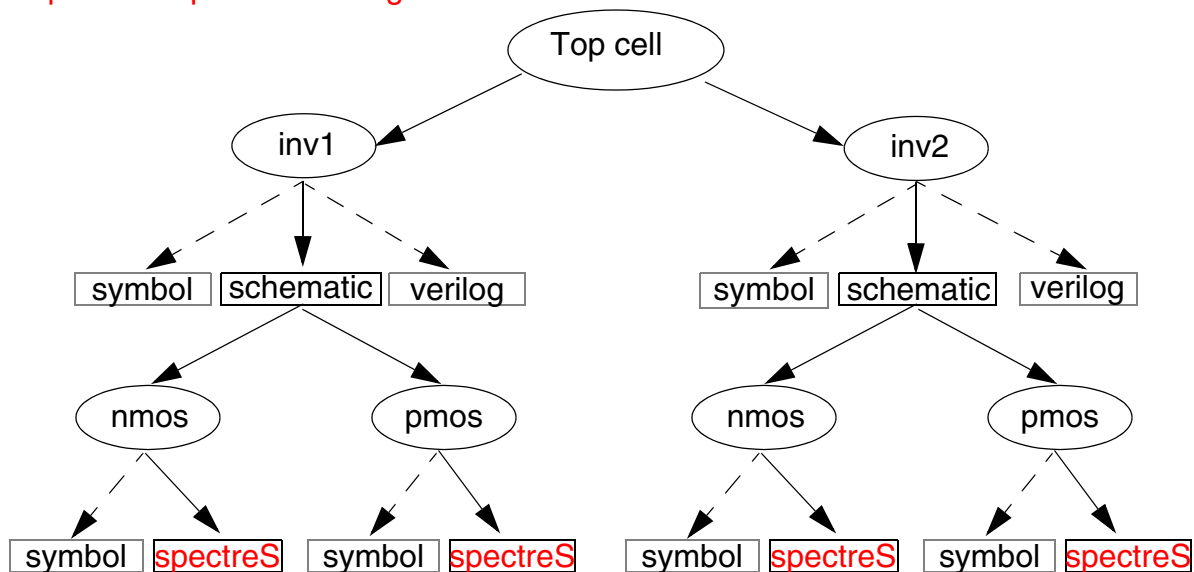
For example, in flattening for placement, you can use the stop list to indicate a cellview that is preplaced, even though it contains instances, so that the placement tool leaves it intact.

A stop list at the global level applies to every level of the configuration. Views must be listed in order of preference. For example, if your stop list is

spectreS verilog

the Hierarchy Editor expands the design as shown in the figure below.

stop list: spectreS verilog



To create or change the stop list,

1. In the *Global Bindings* section, click the *Stop List* field.
2. Do one of the following:
 - ☐ To create a new stop list, type the views in the order of preference.

- ❑ To add a view to an existing stop list, type the new view in the appropriate place. The views are searched in the order in which they are listed.
- ❑ To remove a view from the stop list, highlight the view and press `Delete`.

3. Press `Return`.

4. To see the changes in the configuration, click the *Update* icon in the toolbar.

If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.

The *Tree View* and *Table View* change to show the new configuration resulting from the change in the stop list.

5. To save the changes, choose *File – Save* or press `Control-s`.

Related Topics

[View List Building Forms](#)

[Rules Definition at the Global Level](#)

[Defining a View List at the Global Level](#)

[Defining a Library List at the Global Level](#)

[Defining a Constraint List at the Global Level](#)

Defining a Constraint List at the Global Level

The global constraint list determines the constraints that are used for the configuration. Constraints are design rules that are to be followed during physical implementation. Constraints are stored in views.

The constraint list is a list of constraint views, listed in order of preference. The first view that is found is used. If none of the constraint views in the list are found, the schematic editor creates an empty view named after the first view in the constraint list.

A constraint list can only be defined at the global level.

To create or change a constraint list at the global level,

1. In the *Global Bindings* section of the Hierarchy Editor, click in the *Constraint List* field.
2. Do one of the following:
 - ☐ To create a new constraint list, type the constraint views in order of preference.
 - ☐ To add a constraint view to an existing constraint list, type the new view name in the appropriate place. The views are searched for in the order in which they are listed.
 - ☐ To remove a constraint view from the constraint list, select the view name and press `Delete`.
3. Press `Return`.
4. To see the changes in the configuration, click the *Update* icon in the toolbar.

If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.
5. To save the changes, choose *File – Save* or press `Control-s`.

If you want to specify constraints for a cell, instance, or occurrence, you can create a sub-configuration for that object and specify a constraint list for the sub-configuration.

Related Topics

[Changing a Constraint View Name List](#)

[Configure Physical Hierarchy Window](#)

[Propagation of Constraints Between Schematics and Layout](#)

[Rules Definition at the Global Level](#)

Virtuoso Hierarchy Editor User Guide

Design Components in Hierarchy Editor

Defining a View List at the Global Level

Defining a Stop List at the Global Level

Rules Definition at the Cell Level

You can define rules at the cell level that override the global rules for an individual cell. These rules apply to all instantiations of the cell. In addition, some of these rules can also apply to the hierarchy below the cell.

You specify cell-level rules, also referred to as cell bindings, in the *Cell Bindings* section of the Hierarchy Editor. You must specify cell binding rules in the table view of the Hierarchy Editor; you cannot specify them in the tree view.

You can specify the following rules at the cell level:

- Library Binding
- View Binding
- Stop Point
- Bind-to-Open

Related Topics

[Changing Library Bindings on a Per-Cell Basis](#)

[Changing View Bindings on a Per-Cell Basis](#)

[Defining Stop Points on a Per-Cell Basis](#)

[Defining Bind-to-Open on a Per Cell Basis](#)

[Saving Cell Bindings Table Data to a Text File](#)

Changing Library Bindings on a Per-Cell Basis

You can define a cell-level library list that overrides the global library list for an individual cell. The library list determines the library from which the specific cell is obtained. Libraries are listed in the order of preference. The Hierarchy Editor searches the libraries in the order in which they are listed and uses the cell from the first library in which it is found.

The cell-level library list:

- Applies to every instantiation of the cell
- Is inherited by all components in the hierarchy below the cell

Cell-level library lists can be overridden at the instance and occurrence levels.

Creating a Cell-Level Library List

To create a cell-level library list,

1. Choose *View – Parts Table*.
2. In the *Cell Bindings* section, click in the *Inherited Lib List* column of the cell whose binding you want to change.

The field becomes editable. If the field does not become editable, the library is fixed for your design and cannot be changed.
3. Edit the inherited library list. The library list that is currently displayed is the global library list. Type the libraries you want to use in the order of preference.
4. Press `Return`.
5. To view the changes in the configuration, click the *Update* icon.

If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.

6. To save the changes, choose *File – Save* or press `Control-s`.

In the *Instance Bindings* table, you can see the changes applied to the instances the cell contains. If an instance has an instance-level library list, the cell-level list does not apply to that instance.

Related Topics

[Rules Definition at the Cell Level](#)

[Changing View Bindings on a Per-Cell Basis](#)

[Defining Stop Points on a Per-Cell Basis](#)

[Defining Bind-to-Open on a Per Cell Basis](#)

[Saving Cell Bindings Table Data to a Text File](#)

Changing View Bindings on a Per-Cell Basis

You can define a cell-level view binding that overrides the global view list for an individual cell. Cell-level view binding applies to every instantiation of the cell.

You can either change the binding for only the cell you specify or you can change it in a way that it is also inherited by the components below the cell in the hierarchy. To make the cell binding apply only to the cell, you specify the view in the *View to Use* column. To make the cell binding applicable to components below the cell, you specify a view or view list in the *Inherited View List* column.

To change a binding for a cell on a per cell basis,

1. In the *Cell Bindings* table, select the cell whose bindings you want to change.

2. Do one of the following:

- ☐ Right-click the cell and, from the pop-up menu that appears, select *Set Cell View – newViewName*.

The *Source File* and *Reference Verilog* options on the list of available views let you specify a text file. For more information, see [Using Text Files in Configuration](#).

- ☐ Click in the *View to Use* column, type the new view for the cell, and press `Return`.

The new view name appears in the *View to Use* column in the color used to display user bindings.

- ☐ To remove a binding, right-click the cell and, from the pop-up menu that appears, select *Set Cell View – <none>*.

3. To see the results of your changes, click the *Update* icon in the toolbar.

If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.

4. To save the changes, from the menu bar, click *File – Save* or press `Control-s`.

This new cell binding is not inheritable.

Specifying Cell Binding for a Cell and the Related Objects

To specify a cell binding that applies to the cell as well as to objects below it in the hierarchy,

1. In the *Inherited View List* field, type the new view or list of views for the cell and press `Return`.

2. To see the results of your changes, click the *Update* icon in the toolbar or, from the menu bar, choose *View – Update*.

If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.

The view list you entered is now the view list used for that cell and for any other cells below that cell in the hierarchy (unless they are overridden by other cell or instance bindings lower in the hierarchy).

3. To save the changes, from the menu bar, click *File – Save* or press `Control-s`.

Related Topics

[Changing Library Bindings on a Per-Cell Basis](#)

[Rules Definition at the Cell Level](#)

[Defining Stop Points on a Per-Cell Basis](#)

[Defining Bind-to-Open on a Per Cell Basis](#)

[Saving Cell Bindings Table Data to a Text File](#)

Defining Stop Points on a Per-Cell Basis

In addition to a global stop list that applies to the entire configuration, you can specify stop points on individual cells. A stop point on a cell prevents the cell from being expanded when the hierarchy is expanded.

A stop point on a cell applies to all the instantiations of the cell.

To add a stop point to a cell,

1. Choose *View – Parts Table*.
2. In the *Cell Bindings* section, right-click the cell to which you want to add a stop point.
3. From the pop-up menu, select *Add Stop Point*.

The stop icon appears in the *Info* column of the cell row. All the instantiations of the cell now be considered leaf nodes in the hierarchy; none of them are expanded when the hierarchy is traversed.

Viewing the Cell Instantiations

To see a list of the instantiations to which the stop point applies,

1. In the *Cell Bindings* section, right-click the cell.
2. From the pop-up menu, select *Explain*.

The Explain dialog box appears. The Instantiations section lists the instantiations of the cell. The stop point applies to all these instantiations.

Related Topics

[Changing Library Bindings on a Per-Cell Basis](#)

[Changing View Bindings on a Per-Cell Basis](#)

[Rules Definition at the Cell Level](#)

[Defining Bind-to-Open on a Per Cell Basis](#)

[Saving Cell Bindings Table Data to a Text File](#)

Defining Bind-to-Open on a Per Cell Basis

You can specify that a single instantiation of a cell is to be skipped by setting a bind-to-open attribute on it.

Only non-text instances, such as schematic instances can be skipped using bind-to-open. Bind-to-open should not be set on hierarchical text instances.

You can set the attribute on a cell in two ways:

- Bind-to-open on a cell
- Bind-to-open on a cell in a specific library

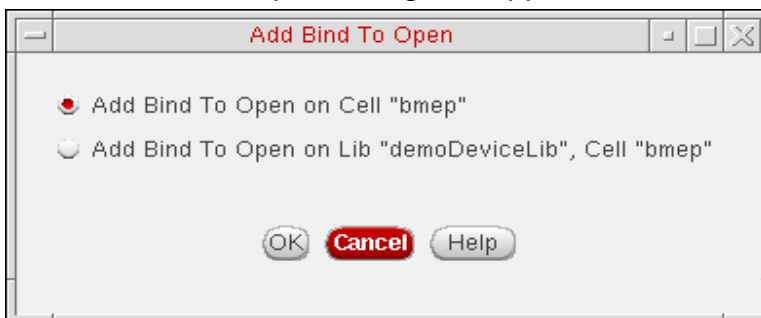
The Hierarchy Editor displays the string ****UNBOUND**** to indicate that a cell has a bind-to-open attribute. ****UNBOUND**** is not an error, unlike ****NONE****. ****UNBOUND**** indicates that the cell is deliberately unbound while ****NONE**** indicates that the binding for the cell could not be determined from the binding rules.

To set a bind-to-open attribute on a cell,

1. Choose *View – Parts Table*.
2. In the *Cell Bindings* section, right-click the cell to which you want to add a bind-to-open attribute.
3. From the pop-up menu, select *Add Bind To Open (Skip Cell)*.

In case multiple rows are selected, the pop-up menu displays the *Add Bind To Open (Skip Multiple Cells)* option.

The Add Bind To Open dialog box appears.



4. In the Add Bind To Open dialog box, do one of the following:
 - ❑ If you want the cell to be unbound, regardless of which library it comes from, select *Add Bind To Open on Cell "cellname"*.

- ❑ If you want the cell to be unbound when it is obtained from a specific library, select *Add Bind To Open on Lib "libname", Cell "cellname"*.

5. Click *OK*.

The cell is now unbound. The *Library*, *View Found* and *View to Use* columns in the Cell Bindings section display ****UNBOUND****. If you selected *Add Bind to Open on Library "libraryname", Cell "cellname"*, only the *View Found* and *View to Use* columns display ****UNBOUND****. Also, the instances contained in the cell are no longer displayed in the Instance Bindings section because the cell is now unbound.

Viewing Instantiations

To see a list of the instantiations to which the bind-to-open attribute applies,

1. In the *Cell Bindings* section, right-click the cell.
2. From the pop-up menu, select *Explain*.

The Explain dialog box appears. The Instantiations section lists the instantiations of the cell. The bind-to-open attribute applies to all these instantiations.

Related Topics

[Rules Definition at the Cell Level](#)

[Defining Stop Points on a Per-Cell Basis](#)

[View List Building Forms](#)

[Changing the Views in the View Choices List Box](#)

[Wildcards in a View List](#)

Rules Definition at the Instance Level

You can define rules at the instance level that override global rules and cell rules. Instance rules apply to only one instantiation of a cell, unlike cell-level rules that apply to all the instantiations of a cell. Note, however, that if the cell that contains the instance is used in multiple places in the design, the binding applies to the instance in all those locations.

Some instance-level rules can also apply to the hierarchy below the instance.

You specify instance-level rules, also referred to as instance bindings, in either of the following places:

- The *Instance Bindings* section in the Table View.
- The Tree View.

In the tree view, any commands you enter (bindings, properties, stop points) are applied to the selected instance by default. If you want to apply the commands to an occurrence, select the *Occurrence Mode* button on the toolbar.

You can specify the following rules at the instance level:

- Library Binding
- View Binding
- Stop Point
- Bind-to-Open

Instance-level rules can be overridden at the occurrence level.

Related Topics

[Changing View Bindings on a Per-Instance Basis](#)

[Changing Library Bindings on a Per-Instance Basis](#)

[Defining Stop Points on a Per-Instance Basis](#)

[Defining Bind-to-Open on a Per-Instance Basis](#)

[Defining Stop Points on a Per-Instance Basis](#)

Changing View Bindings on a Per-Instance Basis

You can specify view bindings for a single instantiation of a cell. You can make the view binding apply to only the instance you specify (effective no lower in the hierarchy) or you can change it in a way that it is also inherited by the components below the instance in the hierarchy.

To make the binding apply only to the instance, you specify the view in the *View to Use* column. To make the binding applicable to components below the instance, you specify a view or view list in the *Inherited View List* column.

To display all of the instances for a particular cell,

1. In the *Cell Bindings* table, click the name of a cell.
2. From the toolbar, click the *Instance Table* button.

The Hierarchy Editor displays the Instance Bindings table.

To specify an instance binding that applies only to the instance,

1. Display either the tree view or the table view of the Hierarchy Editor.
2. If you are displaying the tree view, verify that the *Occurrence Mode* button is deselected.

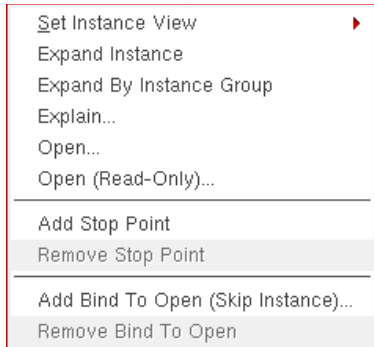
The tree view now displays *Target: Instance*.

3. Do one of the following:
 - a. Click in the *View to Use* column of the instance you want to change and type the new view name.

or,

- a. Right-click the instance you want to change.

The following pop-up menu appears.



- b. Select *Set Instance View* and, from the list of available views, select the view you want.

The *Source File* and *Reference Verilog* options let you specify a text file. For more information, see [Using Text Files in Configuration](#).

The new view name appears in the *View to Use* column in the color used to display user bindings. This binding is not inheritable.

4. To see the results of your changes, click the *Update* icon in the toolbar.

If you selected the *Automatic Update* option in the Options form, this step is not required because your configuration is automatically updated.

5. To save the changes, choose *File – Save* or press `Control-s`.

To specify an instance binding that applies to the instance as well as to objects below it in the hierarchy,

1. Display either the tree view or the table view of the Hierarchy Editor.
2. If you are displaying the tree view, verify that the *Occurrence Mode* button is deselected.

The tree view now displays *Target: Instance*.

3. Click in the *Inherited View List* column of the instance you want to change, type a new view or list of views, and press `Return`.
4. To see the results of your changes, click the *Update* icon in the toolbar.

If you selected the *Automatic Update* option in the Options form, this step is not required because your configuration is automatically updated.

5. To save the changes, from the menu bar, click *File – Save* or press `Control-s`.

Related Topics

[Rules Definition at the Instance Level](#)

[Changing Library Bindings on a Per-Instance Basis](#)

[Defining Stop Points on a Per-Instance Basis](#)

[Defining Bind-to-Open on a Per-Instance Basis](#)

[Defining Stop Points on a Per-Instance Basis](#)

Changing Library Bindings on a Per-Instance Basis

You can define an instance-level library list that overrides the global and cell library lists for a single instantiation of a cell. The library list determines the library from which the instance is obtained. Libraries are listed in order of preference—if the instance is found in the first library in the list, it is used; if it is not found, the second library is searched, and so on.

The instance-level library list:

- Applies to only one instantiation of a cell
- Can apply to multiple objects—if the cell that contains the instance is used in multiple places in the design, the library binding applies to the instance in all those locations
- Is inherited by all components in the hierarchy below the instance

An instance level library binding can be overridden by an occurrence binding.

To create an instance-level library list,

1. Choose *View – Parts Table*.
2. In the *Cell Bindings* table, select the cell that contains the instance you want to change.
3. In the *Instance Bindings* section, click in the *Inherited Lib List* column of the instance whose binding you want to change.

The field becomes editable. If the field does not become editable, the library is fixed for your design and cannot be changed.

4. Edit the inherited library list. The library list that is currently displayed is the library list inherited from higher levels of the hierarchy. Type the libraries in order of preference.
5. Press `Return`.
6. To view the changes in the configuration, click the *Update* icon in the toolbar.

If you selected the *Automatic Update* option in the Options form, this step is not required because your configuration is automatically updated.

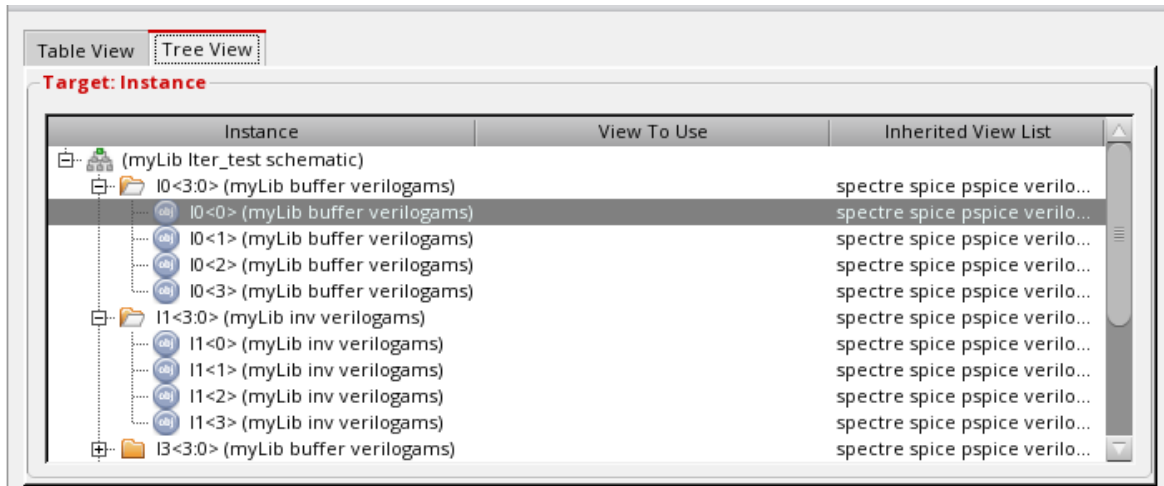
7. To save the changes, choose *File – Save* or press `Control-s`.

To set view bindings separately for individual bits of an iterated instance,

Virtuoso Hierarchy Editor User Guide

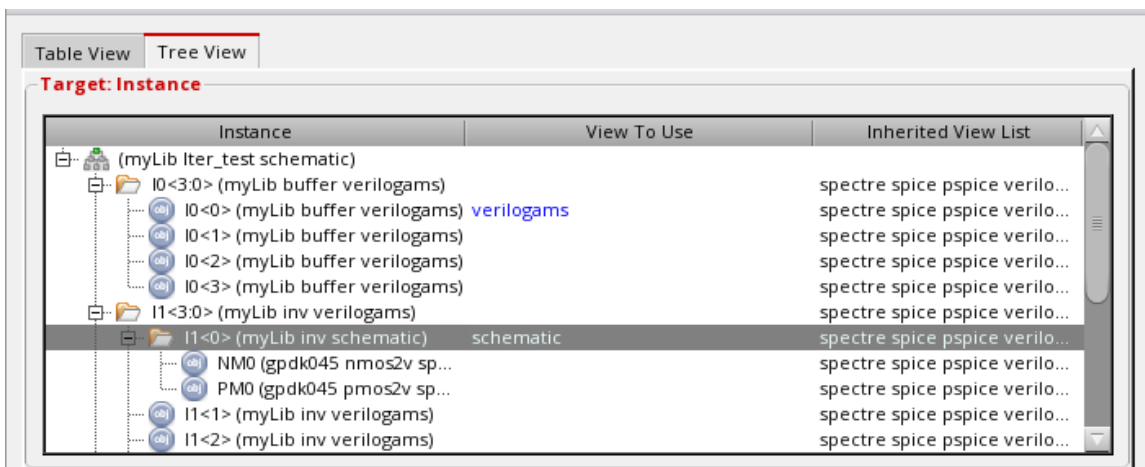
Design Components in Hierarchy Editor

1. Open the view in Hierarchy Editor and switch to the Tree View.



2. Expand an iterated instance by clicking the + icon preceding it.
3. Right-click each bit for which you want to change the view bindings and choose *Set Instance View – <view name>*.

The *View To Use* column in the Tree View is updated with the new binding information.



In the preceding screenshot, iterated instance bit I1<0> is bound to the schematic view whereas the rest of the bits of iterated instance I1<3:0> are bound to verilogams.

4. Click *Recompute the hierarchy* on the toolbar.

You can similarly use the Table View to set iterated instance bit bindings.

Iterated instance bit bindings are currently only supported by the AMS UNL netlist.

Related Topics

[Rules Definition at the Instance Level](#)

[Changing Library Bindings on a Per-Instance Basis](#)

[Defining Stop Points on a Per-Instance Basis](#)

[Defining Bind-to-Open on a Per-Instance Basis](#)

[Defining Stop Points on a Per-Instance Basis](#)

Defining Stop Points on a Per-Instance Basis

You can specify a stop point on a single instantiation of a cell that prevents the instance from being expanded when the hierarchy is expanded.

A stop point on an instance can apply to multiple objects—if the cell that contains the instance is used in multiple places in the design, the stop point applies to the instance in all those places. For example, if you put a stop point on instance `I2` of cell `OpAmp` and cell `OpAmp` is used in three places in the design, the stop point applies to instance `I2` in all three instantiations of `OpAmp`.

You can add a stop point to an instance from either the tree view or the table view of the Hierarchy Editor.

To add a stop point to an instance from the tree view,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. If the *Occurrence Mode* button on the toolbar is selected, deselect it.

The tree view now displays *Target: Instance*.

3. Right-click the instance on which you want to add a stop point.
4. From the pop-up menu, select *Add Stop Point*.

The instance icon changes to a stop point icon. The instance can no longer be expanded.

If the instance has already been identified as an occurrence and the tree view displays *Target: Occurrence* when the instance is selected, the stop point is added to the occurrence.

To add a stop point to an instance from the table view,

1. Choose *View – Parts Table* and *View – Instance Table* to display the table view of the Hierarchy Editor, if it is not already displayed.
2. In the *Cell Bindings* section, select the cell that contains the instance to which you want to add a stop point.

The *Instance Bindings* section displays the instances contained in the cell.

3. In the *Instance Bindings* section, right-click the instance to which you want to add a stop point.
4. From the pop-up menu, select *Add Stop Point*.

The stop point icon appears in the *Info* column of the instance row. The instance can no longer be expanded.

If the cellview that contains the instance is used in multiple places in the design, the stop point applies to the instance in all those places.

To see which objects the stop point applies to,

1. In the *Cell Bindings* section, right-click the cell that contains the instance to which you added the stop point.
2. From the pop-up menu, select *Explain*.

The Explain dialog box appears. The *Instantiations* section lists the instantiations of the cell. The stop point applies to the instance in all these instantiations of the cell.

You cannot add an instance stop point to an object that has already been defined as an occurrence. When you add a stop point to such an object, it is added to the occurrence, not the instance, and applies only to one object at a specific path.

Related Topics

[Rules Definition at the Instance Level](#)

[Defining Bind-to-Open on a Per-Instance Basis](#)

[Defining Stop Points on a Per-Instance Basis](#)

Defining Bind-to-Open on a Per-Instance Basis

You can specify that a single instantiation of a cell is to be skipped by setting a bind-to-open attribute on it.

Only non-text instances, such as schematic instances can be skipped using bind-to-open. Bind-to-open should not be set on hierarchical text instances.

A bind-to-open attribute on an instance can apply to multiple objects—if the cell that contains the instance is used in multiple places in the design, the bind-to-open applies to the instance in all those places. For example, if you specify that instance `I2` in cell `OpAmp` is unbound and cell `OpAmp` is used in three places in the design, `I2` gets unbound in all three instantiations of `OpAmp`.

The Hierarchy Editor displays the string `**UNBOUND**` to indicate that an instance has a bind-to-open attribute. `**UNBOUND**` is not an error, unlike `**NONE**`. `**UNBOUND**` indicates that the instance is deliberately unbound while `**NONE**` indicates that the binding for the instance could not be determined from the binding rules.

You can add a bind-to-open attribute to an instance from either the tree view or the table view of the Hierarchy Editor.

To add a bind-to-open attribute to an instance from the tree view,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. If the *Occurrence Mode* button on the toolbar is selected, deselect it.
The tree view now displays *Target: Instance*.
3. Right-click the instance on which you want to add a bind-to-open.
4. From the pop-up menu, select *Add Bind To Open (Skip Instance)*.

In case multiple rows are selected, the pop-up menu displays the *Add Bind To Open (Skip Multiple)* option.

The instance is now unbound. The library, cell, and view name of the instance are replaced by `**UNBOUND**`. For example:

 Q10 (**UNBOUND** **UNBOUND** **UNBOUND**) ****UNBOUND**** spectreS cdsSpice spice

If the instance has already been identified as an occurrence and the tree view displays *Target: Occurrence* when the instance is selected, the bind-to-open attribute is applied to the occurrence.

Virtuoso Hierarchy Editor User Guide

Design Components in Hierarchy Editor

To add a bind-to-open attribute to an instance from the table view,

1. Choose *View – Parts Table* and *View – Instance Table* to display the table view of the configuration, if it is not already displayed.
2. In the *Cell Bindings* section, select the cell that contains the instance to which you want to add a bind-to-open attribute.

The *Instance Bindings* section displays the instances contained in the cell.

3. In the *Instance Bindings* section, right-click the instance to which you want to add a bind-to-open.
4. From the pop-up menu, select *Add Bind To Open (Skip Instance)*.

The instance is now unbound. The *Library*, *Cell*, *View Found*, and *View to Use* columns display ****UNBOUND****. For example:

Instance Bindings (tdmsDemoLib opamp schematic)				
Instance	Library	Cell	View Found	View To Use
C74	**UNBOUND**	**UNBOUND**	**UNBOUND**	**UNBOUND**

If the cellview that contains the instance is used in multiple places in the design, the bind-to-open applies to the instance in all those places.

To see which objects the bind-to-open applies to,

1. In the *Cell Bindings* section, right-click the cell that contains the instance to which you added the bind-to-open.
2. From the pop-up menu, select *Explain*.

The Explain dialog box appears. The *Instantiations* section lists the instantiations of the cell. The bind-to-open applies to the instance in all these instantiations of the cell. If any of the instances has an occurrence binding on it already, the bind-to-open does not apply to it.

Related Topics

[Rules Definition at the Instance Level](#)

[Defining Occurrence-Level Bind-to-Open](#)

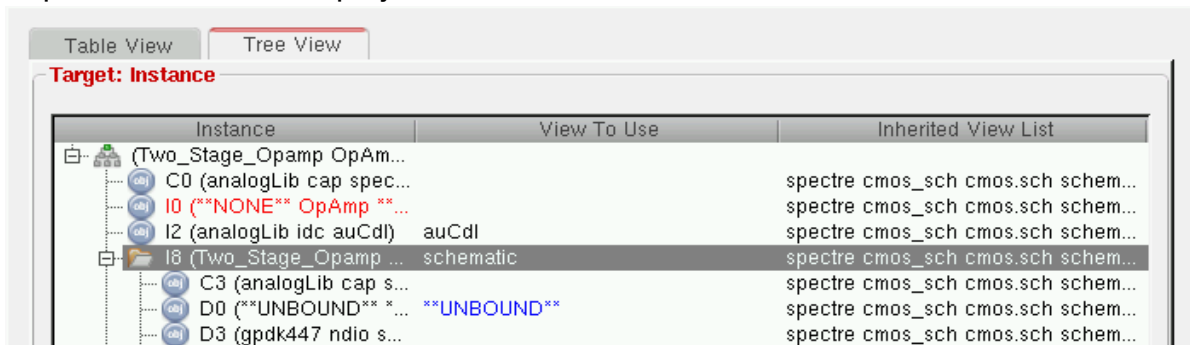
Changing Instance Bindings Inside a Block

A block is a group of instances used in simulators like VHDL. Blocks can contain other blocks and other instances. You can bind instances inside a block the same way you can bind instances in a cellview.

Blocks are better displayed in the tree structure of the Hierarchy Editor.

To change the instance bindings inside a block,

1. Select the *Tree View* tab to display the tree view.
2. Select the block containing the instance you want to change.
3. Expand the block to display the instances within it.



4. Right-click the instance whose bindings you want to change.

The pop-up menu appears.

5. From the pop-up menu, select *Set Instance View – newView*.

The new view name appears in the *View to Use* column in the color used to display user bindings.

6. To see the results of your changes, click the *Update* icon in the toolbar.

If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.

7. To save the changes, from the menu bar, click *File – Save* or press `Control-s`.

Related Topics

[Rules Definition at the Instance Level](#)

[Changing View Bindings on a Per-Instance Basis](#)

Rules Definition at the Occurrence Level

An occurrence is an instance that is uniquely identified by its full path from the top-level design to the instance. Occurrences provide the ability to uniquely identify an object in the design and assign attributes to that object.

Occurrence binding is particularly useful for mixed-signal simulation.

In the Hierarchy Editor, you can assign the following attributes to an occurrence:

- Occurrence binding
- Occurrence stop point
- Occurrence-level bind-to-open





Setting any of the above attributes identifies the object as an occurrence.

The Hierarchy Editor provides an occurrence editing mode in the tree view. You can turn on the mode by selecting the following button in the toolbar:



The tree view displays *Target: Occurrence* when you select any object in the tree to indicate that any change you make gets applied to the occurrence.

In the Hierarchy Editor, occurrences are represented by the following icons:

-  The object has an occurrence binding
-  The tree node contains an occurrence
-  The object has an occurrence stop point
-  The object has an occurrence binding and an occurrence stop point

To see a complete list of icons and color conventions used in the Virtuoso Hierarchy Editor,

- ➡ Choose *Help – Legend*.

The Legend dialog box appears. This dialog box describes all the icons and color conventions.

Not all Cadence tools support occurrences. Before you use the occurrence feature of the Hierarchy Editor, refer to the documentation for the applications you are using to check if those applications support occurrences.

Related Topics

[Difference between Occurrences and Instances](#)

[Defining Occurrence Bindings](#)

[Defining Occurrence Stop Points](#)

[Defining Occurrence-Level Bind-to-Open](#)

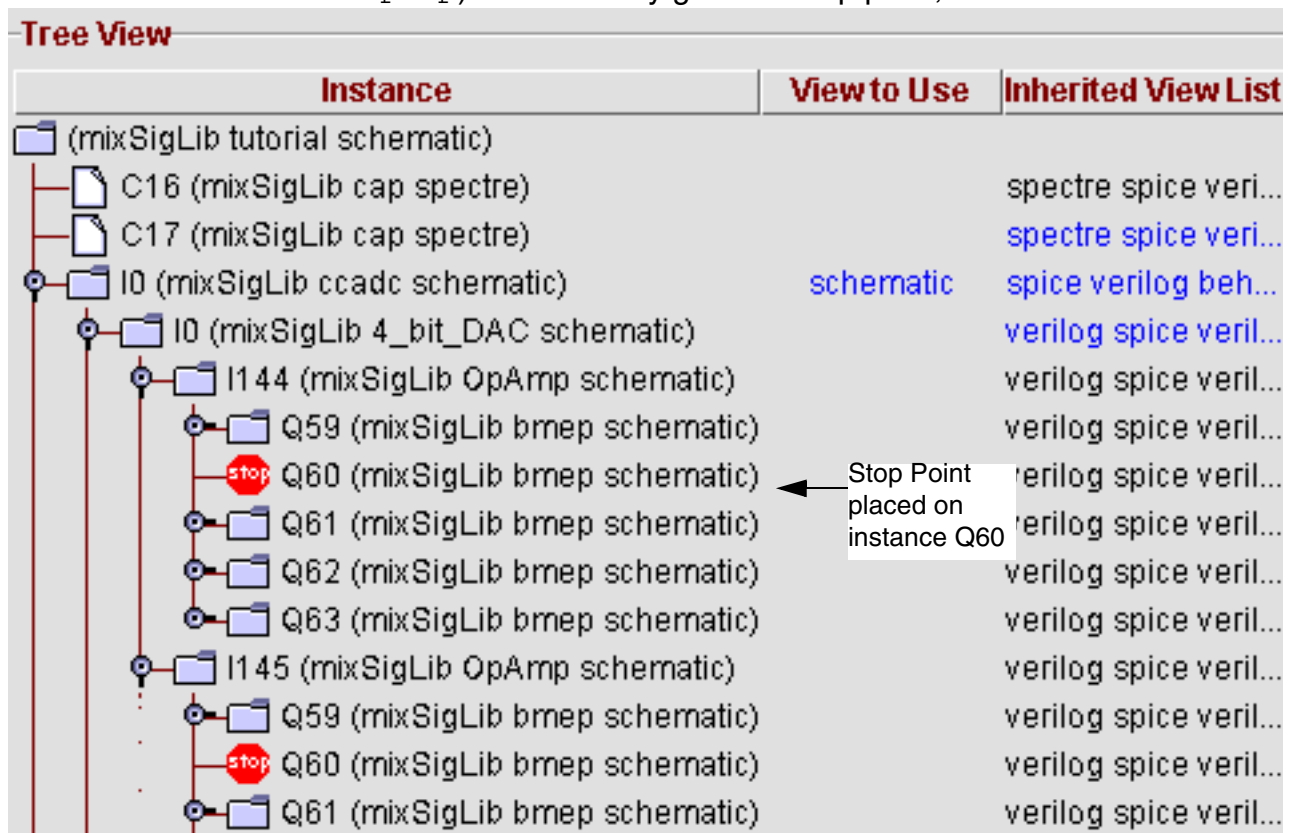
Difference between Occurrences and Instances

Occurrences are unique, while instances are not. When you put an attribute (such as a stop point) on an instance, it can apply to more than one object because instances cannot be identified uniquely. An instance is identified by the cell it is contained in. If the cell is used in multiple places in the design, the instance is in multiple places in the design.

An occurrence, on the other hand, is identified uniquely by its full path from the top-level design. When you put an attribute (such as a stop point) on an occurrence, it only applies to that single object.

The following example illustrates the difference between instances and occurrences. Both Figure 1 and Figure 2 display the design `mixSigLib.tutorial:schematic`. In this design, the cell `OpAmp` is used twice—its instantiations are `mixSigLib.tutorial:schematic.I0.I0.I144` and `mixSigLib.tutorial:schematic.I0.I0.I145`. The cell `OpAmp` contains several instances, including `Q60`.

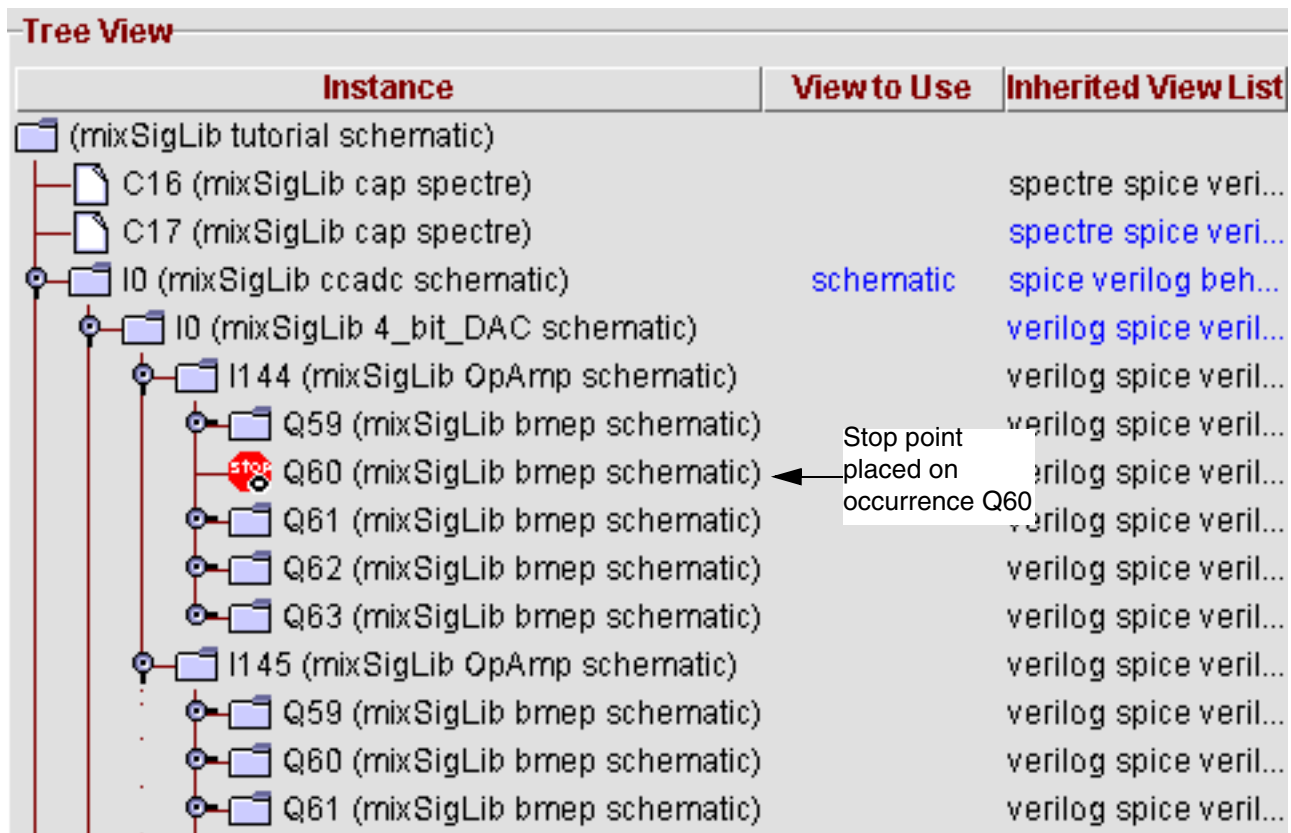
In Figure 1, an instance stop point is specified on `Q60` in `I144`. Notice that `Q60` in `I145` (the other instantiation of cell `OpAmp`) automatically gets the stop point, too.



Virtuoso Hierarchy Editor User Guide

Design Components in Hierarchy Editor

In Figure 2, an occurrence stop point is specified on Q60 in I144. Notice that Q60 in I145 (the other instantiation of cell OpAmp) is not affected by the stop point. This is because the stop point was put on the occurrence and not on the instance.



Related Topics

[Rules Definition at the Occurrence Level](#)

[Defining Occurrence Bindings](#)

[Defining Occurrence Stop Points](#)

[Defining Occurrence-Level Bind-to-Open](#)

Defining Occurrence Bindings

You create an occurrence binding by specifying a library, cell, and view—or a text file—for a single object at a specific path in the design. Unlike cell or instance binding, occurrence binding applies only to one object.

When you do an occurrence binding, none of the other binding rules—such as inherited library lists or view lists—apply to that occurrence.

You can create an occurrence binding only in the tree view of the Hierarchy Editor, in the occurrence editing mode; you cannot create it in the table view.

If the library or cell is fixed by some other rules in the design data, you will not be able to change that binding and will only be able to set the view. For example, design data created by the Virtuoso schematic editor typically does not allow you to change the library and cell.

Creating an Occurrence Binding

You can bind an occurrence to

- A library, cell, and view

If the library and cell are fixed by some other rules in the design data, you can only set the view.

- A source file
- A Verilog file

This section describes how to bind an occurrence to a library/cell/view. For information on how to bind an occurrence to a source file or a Verilog file, see [Using Text Files in Configuration](#).

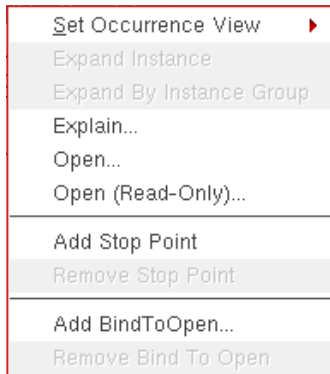
To specify a library/cell/view binding for an occurrence,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. Turn on the occurrence editing mode by clicking the Set bindings...on Occurrences button in the toolbar:

The tree view displays *Target: Occurrence* as the section header.

3. Right-click the object to which you want to add an occurrence binding.




The following pop-up menu appears:



4. From the pop-up menu, Select *Set Occurrence View* and select a view from the list of available views.

The Hierarchy Editor sets the view binding to the view you specify and the library and cell binding to the current library and cell.

You can also set the view by typing it in the *View to Use* column of the occurrence.

The object is now identified as an occurrence and a library/cell/view binding is set on it. The leaf node or tree node icon that was displayed for the object is replaced by the  icon (occurrence icon). If the object already had an occurrence stop point, the  icon is replaced by the  icon (occurrence binding and stop point icon).

Editing an Occurrence Binding

To edit an occurrence binding,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. Turn on the occurrence editing mode by clicking the following button in the toolbar:

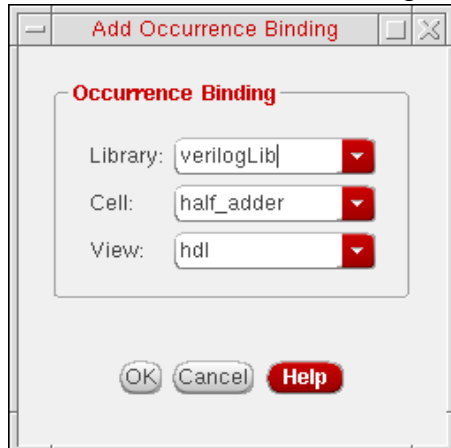
The tree view displays *Target: Occurrence*.

3. Select the occurrence whose binding you want to change.
4. If you want to change only the view, do one of the following:
 - ☐ Right-click the occurrence, select *Set Occurrence View* on the pop-up menu, and then select the new view from the list of available views.
 - ☐ Type the new view in the *View to Use* column of the selected occurrence.

Skip the remaining steps in this section.

5. If you want to change the library or cell,
 - a. Right-click the occurrence.
 - b. From the pop-up menu that appears, select *Set Occurrence View – Library/Cell/View*.

The Add Occurrence Binding dialog box appears.



If the library and cell are fixed for the design data, then the *Set Occurrence View – Library/Cell/View* option is not available. You can only specify a view binding. For example, Virtuoso design data typically does not allow you to change the library and cell.

- c. Edit the library, cell, and view to which the occurrence is bound.
- d. Click *OK*.

The dialog box closes.

To edit an occurrence binding to a source file,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. Turn on the occurrence editing mode by clicking the *Set bindings...on occurrence* button in the toolbar:

The tree view displays *Target: Occurrence*.

3. Right-click the occurrence whose binding you want to change.
4. From the pop-up menu, select *Set Occurrence View – Source File*.

The Enter the sourcefile Location form appears. For information on specifying a source file, see [Using Source Files in Configuration](#).

To edit an occurrence binding to a Verilog file,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. Turn on the occurrence editing mode by clicking the *Set bindings...on occurrence* button in the toolbar:

The tree view displays *Target: Occurrence*.

3. Right-click the occurrence whose binding you want to change.
4. From the pop-up menu, select *Set Occurrence View – Reference Verilog*.

The Reference Verilog Modules form appears. For information on specifying a Verilog file, see [Referencing a Verilog File](#).


Removing an Occurrence Binding

To remove an occurrence binding,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. Turn on the occurrence editing mode by clicking the *Set bindings...on occurrence* button in the toolbar:

The tree view displays *Target: Occurrence*.

3. Right-click the occurrence whose binding you want to remove.
4. From the pop-up menu, select *Set Occurrence View – <none>*.

The occurrence binding is removed from the object. The  icon (occurrence icon) is replaced by the leaf or folder icon. The object is no longer uniquely identified, unless it also has an occurrence stop on it.

Related Topics

[Add Occurrence Binding Form](#)

Defining Occurrence Stop Points

An occurrence stop point is a stop point on a specific path and applies only to one object in the design.

If an object has already been defined as an occurrence (an object is defined as an occurrence if it has one of the following occurrence attributes: occurrence binding, occurrence stop point, or occurrence bind-to-open), when you add a stop point, you are automatically adding it to the occurrence, not to the instance.

You can only add an occurrence stop point in the tree view of the Hierarchy Editor; you cannot add it in the table view.


Adding an Occurrence Stop Point

To add an occurrence stop point,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. Turn on the occurrence editing mode by clicking the *Set bindings...on occurrence* button in the toolbar:

The tree view displays *Target: Occurrence*.

3. Right-click the object on which you want to add a stop point.
4. From the pop-up menu, select *Add Stop Point*.

The  icon (occurrence stop point icon) appears next to the occurrence. The occurrence cannot be expanded until the stop point is removed.

Removing an Occurrence Stop Point

To remove an occurrence stop point,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. Right-click the occurrence.
3. From the pop-up menu, select *Remove Stop Point*.

The occurrence stop point is removed. The  icon is replaced by the leaf or folder icon. The object is no longer uniquely identified, unless it also has an occurrence binding.

Related Topics

[Rules Definition at the Occurrence Level](#)

[Difference between Occurrences and Instances](#)

Defining Occurrence-Level Bind-to-Open

You can specify that a single instantiation of a cell is to be skipped by setting a bind-to-open attribute on it.

Only non-text instances, such as schematic instances can be skipped using bind-to-open. Bind-to-open should not be set on hierarchical text instances.

The Hierarchy Editor displays the string ****UNBOUND**** to indicate that an occurrence has a bind-to-open attribute. ****UNBOUND**** is not an error, unlike ****NONE****. ****UNBOUND**** indicates that the occurrence is deliberately unbound while ****NONE**** indicates that the binding for the occurrence could not be determined from the binding rules.

You can only add an occurrence-level bind-to-open attribute in the tree view of the Hierarchy Editor, you cannot add it in the table view.

Setting Bind-to-Open on an Occurrence

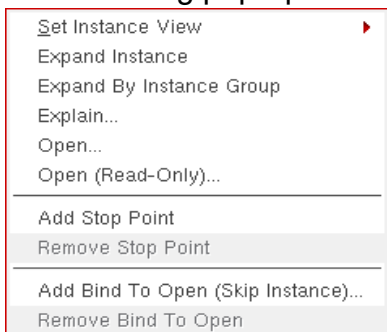
To add a bind-to-open attribute to an occurrence,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. Turn on the occurrence editing mode by clicking the following *Set bindings...on occurrence* in the toolbar:


The tree view displays *Target: Occurrence*.



3. Right-click the object on which you want to add an occurrence-level bind-to-open.

The following pop-up menu appears:



4. From the pop-up menu, select *Add Bind To Open (Skip Instance)*.

The bind-to-open attribute is set on the occurrence. The  icon (occurrence icon) appears next to the occurrence and the library, cell, and view names are replaced by ****UNBOUND****. For example:

  AND2_0 (**UNBOUND** **UNBOUND** **UNBOUND**) ****UNBOUND**** spectreS cds:

Removing Occurrence-Level Bind-to-Open

To remove an occurrence-level bind-to-open attribute,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. Right-click the occurrence from which you want to remove the bind-to-open attribute.
3. From the pop-up menu, select *Remove Bind To Open*.

The occurrence is no longer unbound. The library, cell, and view name are determined based on the other binding rules and displayed.

Related Topics

[Rules Definition at the Occurrence Level](#)

[Difference between Occurrences and Instances](#)

Using Text Files in Configuration

In your design configuration, you can also use text files that contain source descriptions of design elements. Instead of binding a cell, instance, or occurrence to a view, you can specify that a text file be used. These file bindings are similar to view bindings.

You can use the following types of text files in a configuration:

- Source files for HSPICE or SPICE design blocks
- Verilog files

Using Source Files in Configuration

You can use source files for HSPICE or SPICE design blocks in your design configuration. You use a source file by binding a cell, instance, or occurrence to the source file. This file binding is similar to a view binding.

Binding to a source file puts the `sourcefile` property on the cell, instance, or occurrence. The value of the property is the path to the source file and it is stored in the `prop.cfg` file in the configuration view. The `sourcefile` property is used by other applications in the flow.

To bind a cell, instance, or occurrence to a source file,

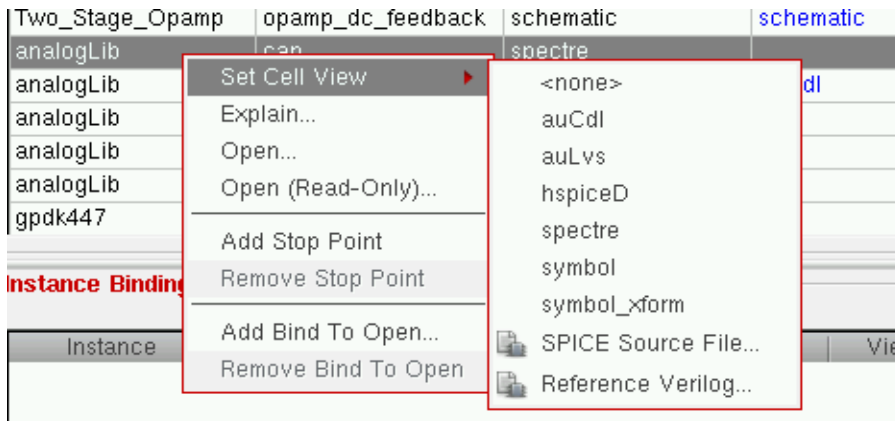
1. If you are binding a cell, display the table view of the Hierarchy Editor; if you are binding an instance, display either the table view or the tree view; if you are binding an occurrence, display the tree view and turn on the occurrence editing mode.
2. Right-click the cell, instance, or occurrence.

Even if you are not in occurrence editing mode, when you edit an object in the tree view that has been identified as an occurrence (by having an occurrence binding, occurrence stop point, or occurrence bind-to-open on it), you are editing the occurrence, not the instance. The tree view displays *Target: Occurrence* when you select such an object.

Virtuoso Hierarchy Editor User Guide


Design Components in Hierarchy Editor

3. From the pop-up menu, select *Set Cell View / Set Instance View / Set Occurrence View – SPICE Source File*. For example:



The Enter the SPICE/HSPICE source file Location form appears.

4. In the form, do one of the following:
 - ☐ Type the path to the source file in the *File Name* field. You can use environment variables in the path.
 - ☐ Click *Browse* and use the Select the source file value form to select the file.
5. Click *OK*.

The cell, instance, or occurrence is bound to the source file. The *View to Use* column displays the path to the file. The  icon next to the path indicates that it is a file binding.

To remove a source file binding,

1. If you are editing a cell, display the table view of the Hierarchy Editor; if you are editing an instance, display either the table view or the tree view; if you are editing an occurrence, display the tree view and turn on the occurrence editing mode.
2. Right-click the cell, instance, or occurrence.
3. From the pop-up menu, select *Set Cell View / Set Instance View / Set Occurrence View – <none>*.

Related topics

Using Verilog Files in Configuration

Using Verilog Files in Configuration

You can use Verilog views in a design configuration in the same way that you use other views. Verilog views contain Verilog text files such as `verilog.v`, `verilog.vams`, or `verilog.va`, which describe Verilog modules.

However, if you have Verilog text files that are not in a Cadence design library, that is, they are not in the Cadence library/cell/view structure, you can still use them in your configuration in the following ways:

- By using the Hierarchy Editor's *Populate Library* command to bring Verilog modules from these text files into a Cadence design library.

The Hierarchy Editor creates cellviews for all the modules in the library you specify. Use this option when you want the cellviews to be created in a master library or an explicit temporary directory.

- By referencing the Verilog text file.

The file is later compiled, by other tools in the flow, into the implicit temporary directory. Use this option when you cannot create cellviews in the master library or explicit temporary directory.

You can bring Verilog modules into your Cadence design library so that you can use them in your design configurations. The Hierarchy Editor creates a cellview in the library for each module, with corresponding `pc.db` and `master.tag` files, as well as a link to the original source file.

You can get modules from multiple Verilog source files at the same time. Each Verilog file can contain a single module or several modules that are independent of each other or hierarchically related.

You do not have to open a configuration to populate a library with Verilog modules. If you do open a configuration, you have the choice of updating the global view list and library list of the configuration with the Verilog views or the library you specify.

To populate a library with Verilog modules,

1. Choose *File – Populate Library – Verilog*.

The Populate Library with Verilog Modules form appears.

2. Click *OK* after specifying all the required details.

The dialog box closes and the library is populated with Verilog modules. If there are any errors, they are displayed in the Messages section of the Hierarchy Editor.

If a configuration is open and you chose to update the library and view lists, the updated lists are displayed in the *Global Bindings* section as well as the *Inherited Library List* and *Inherited View List* columns.

The Verilog views are now available for binding. You can bind a cell, instance, or occurrence to the Verilog views in the same way that you bind to other views.

Compiler Options

The Hierarchy Editor uses the xmvlog compiler to create cellviews. It uses the following options with xmvlog:

```
xmvlog -use5x -work libraryName [-view viewName] [-ams]
```

where `-use5x` specifies that the Cadence library structure be created for the modules; `-work libraryName` specifies the library in which the cellviews are created; `-view viewName`, which specifies the name of the view, is used only if you fill in the *View* field in the Populate Library with Verilog Modules form (the default view name is `module`); and `-ams` is used for any source files that have a `.va` or `.vams` extension. For more information about xmvlog, see the *Cadence NC-Verilog Simulator Help*.

If the AMS plug-in is installed, the Hierarchy Editor runs xmvlog with the options specified in the *AMS – Options – Compiler* form.

If you want the xmvlog compiler to be run with any other options, specify them in the `hdl.var` file. For more information about the `hdl.var` file, see the *Cadence NC-Verilog Simulator Help*.

Referencing a Verilog File

If you want to use Verilog modules from a Verilog text file that is outside the Cadence library/cell/view structure and you cannot create cellviews for these modules in the master or explicit temporary library, you can reference the Verilog file. The file gets compiled later, by other tools in the flow, into the implicit temporary directory.

You reference a Verilog file by

- Specifying the path to the file; and
- Binding the cell, instance, or occurrence to a view that gets implemented later with the modules in the Verilog file

This puts the `verilogfile` property on the cell, instance, or occurrence. The value of the property is the path to the Verilog file and it is stored in the `prop.cfg` file in the configuration

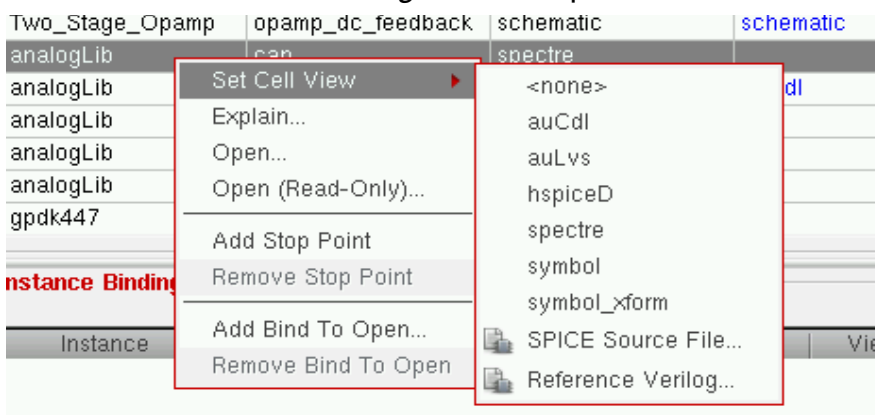
Virtuoso Hierarchy Editor User Guide

Design Components in Hierarchy Editor

view. The property is used by downstream processes such as the design preparation step of AMS, which reads the property and compiles the file it refers to into the implicit temporary directory.


To reference a Verilog file for a cell, instance, or occurrence,

1. For a cell, display the table view of the Hierarchy Editor; for an instance, display either the table view or the tree view; for an occurrence, display the tree view and turn on the occurrence editing mode.
2. Right-click the cell, instance, or occurrence.
3. From the pop-up menu, select *Set Cell View / Set Instance View / Set Occurrence View — Reference Verilog*. For example:



The Reference Verilog Modules form appears.

4. Click *OK* after specifying all the required details.

The cell, instance, or occurrence is bound to the view. The *View to Use* column displays the view name. The  icon (file icon) next to the view name indicates that it gets implemented with a text file.

To remove a Verilog file reference,

1. If you are editing a cell, display the table view of the Hierarchy Editor; if you are editing an instance, display either the table view or the tree view; if you are editing an occurrence, display the tree view and turn on the occurrence editing mode.
2. Right-click the cell, instance, or occurrence.
3. From the pop-up menu, select *Set Cell View / Set Instance View / Set Occurrence View — <none>*.

Related topics

[Using Text Files in Configuration](#)

[Populate Library with Verilog Modules Form](#)

[Reference Verilog Modules Form](#)

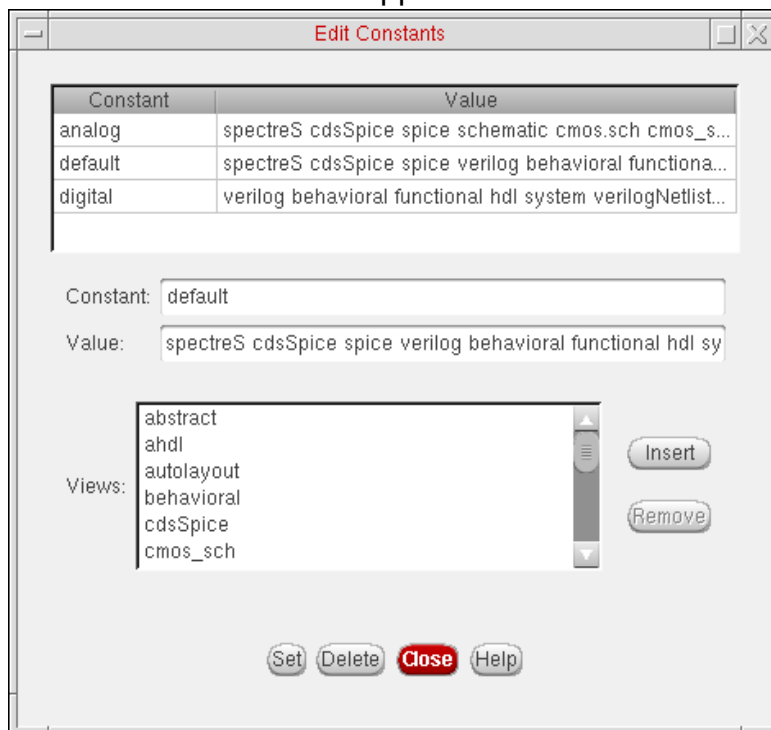
Creating and Editing Constants

A constant is a symbolic name that represents a view list. Constants provide a way to store and retrieve long view lists. You can create a constant and use it anywhere you use a View List and a Stop List.

To create a constant,

1. Choose *Edit – Constants*.

The Edit Constants form appears.



2. In the *Constant* field, type the name you want to use to denote the view list.
3. In the *Value* field, specify the view list by doing one of the following:
 - ☐ Select a view in the *Views* list box, then click *Insert*. Repeat for all the views you want to add to the view list.
 - ☐ Type the view list.

The view names must appear in order of preference. The first view that is found is used.

You can change the default list of views in the *View Choices* list box.

4. Click *Set*.

The Hierarchy Editor creates the constant. The constant and its value appear in the table at the top of the form.

To edit the views included in an existing constant,

1. Choose *Edit – Constants*.

The Edit Constants form appears.

2. In the table at the top of the form, click the constant whose views you want to change.

The constant name is displayed in the *Constant* field and its views are displayed in the *Value* field.

3. To add a view,

- a. In the *Value* field, place the cursor in the position that you want to add the new view.

Views must be listed in order of preference—the first view that is found is used.

- b. In the *Views* list box, select the view that you want to add.

- c. Click *Insert*.

4. To remove a view,

- a. In the *Value* field, double-click the view.

- b. Press the `Delete` key.

5. When you have finished editing the *Value* field, click *Set*.

Related Topics

[Changing the Views in the View Choices List Box](#)

[Edit Constants Form](#)

Changing the Views in the View Choices List Box

The default list of views in the *Views* list box is defined by the `hed.constants` variable in the `your_install_dir/share/cdssetup/hierEditor/env/hed.env` file. You cannot edit this registration file but you can change the list of views by adding the variable to a local `hed.env` file.

To change the views in the *Views* list box,

1. If you do not have a `hed.env` file, create a file named `hed.env` in a directory that is in your `setup.loc` file (for example, your `$HOME` directory or current working directory).

If you have multiple `hed.env` files, the Hierarchy Editor determines which value to use based on the search order defined in the `setup.loc` file.

2. Add the following variable to the `hed.env` file:

```
hed.constants viewChoices string "view1 view2 view3 view4 ..."
```

where `view1`, `view2`, `view3`, and `view4` are the views that you want the *Views* list box to display. For example:

```
hed.constants viewChoices string "abstract ahdl behavioral cdsSpice  
cmos_sch functional schematic spectre spectreS symbol"
```

To use a constant,

- ➔ Type the name of the constant, preceded by a dollar sign (\$), in any view list or stop list field.

By default, the Hierarchy Editor displays the constant in the view lists or stop lists in which it is used. However, you can choose to display the expanded form of the constant (the list of views that the constant represents) instead of displaying the constant name.

To display the expanded form of the constant,

- ➔ Add the following variable to the `hed.env` file:

```
hed.display showConstInViewList boolean nil
```

Related Topics

[viewChoices](#)

[showConstInViewList](#)

Wildcards in a View List

You can use the asterisk character (*) as a wildcard in any view list—global or inherited. Use the following format to specify a wildcard in a view list:

```
[viewName ...] *
```

For example:

```
*
```

```
schematic *
```

```
schematic vhdl *
```

You cannot use * as part of a view name. For example, you cannot have `ns*` or `n*s` as a view in the view list.

The * wildcard matches any view in the cell directory. If there is more than one view, then the view that was most recently modified is picked. To determine the most recently modified view, the Hierarchy Editor compares the timestamps of the master files of each view. (The master file is the file that the `master.tag` file for the view points to.)

Any changes in a view that do not change the master file are not considered. For example, in the AMS Designer flow, properties associated with a cellview can change without affecting the timestamp of the master file of the view.

For example:

View list	View
*	The most recently modified view
schematic *	The <code>schematic</code> view; if the <code>schematic</code> view does not exist, the most recently modified view
schematic vhdl *	The <code>schematic</code> view; if the <code>schematic</code> view does not exist, the <code>vhdl</code> view; if the <code>vhdl</code> view does not exist, the most recently modified view

Related Topics

[Changing the Views in the View Choices List Box](#)

[Library Scoped Views in a View List](#)

Library Scoped Views in a View List

You can use library scoped views in the view list at the global level or in an inherited view list at the cell, instance, or occurrence levels.

To specify a library scoped view, use the `mylib::myview` syntax to indicate that the specified view should only be considered if the specified library matches the library in which a cell is present.

For example, if your view list is:

```
spectreS mylib::schematic cmos.sch verilog
```

the schematic view is only considered as matching when a cell resides in the library mylib.

Related Topics

[Defining a View List at the Global Level](#)

Changing Binding Data Color Definitions

The Hierarchy Editor displays color-coded cell and instance binding data. The following table lists the default colors:

Color	Definition
Black	Default binding
Blue	User-defined binding
Red	Error
Orange	Not in Use

As you type new binding information into the Cell Bindings or Instance Bindings table, the user-defined binding color is used.

You can change the default colors by adding variables to a local `hed.env` file.

To change the default colors,

1. If you do not have a `hed.env` file, create a file named `hed.env` in a directory that is specified in your `setup.loc` file (for example, your `$HOME` directory or current working directory). You can create the file with a text editor or with the *File – Save Defaults* command.

If you have multiple `hed.env` files, the Hierarchy Editor determines which value to use based on the search order defined in the `setup.loc` file.

2. Add the following variables to the `hed.env` file:

```
hed.colorChooser bindingError string "newColor"  
hed.colorChooser userBinding string "newColor"  
hed.colorChooser defaultBinding string "newColor"  
hed.colorChooser notInUse string "newColor"
```

For example, if you want to change the default binding color to green, you would add the following variable:

```
hed.colorChooser defaultBinding string "green"
```

Add only those variables that you want to change from the default.

3. Save the `hed.env` file.

Your changes are displayed the next time you start the Hierarchy Editor.

Related Topics

[bindingError](#)

[defaultBinding](#)

[notInUse](#)

[userBinding](#)

Saving Cell Bindings Table Data to a Text File

You can save data from the *Cell Bindings* table in an ASCII file, which you can print or import into a spreadsheet or other application.

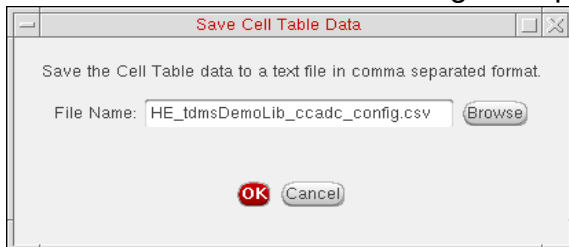
To save the *Cell Bindings* table data to a text file,

1. Customize the *Cell Bindings* table to display only the data that you want to save to a text file:
 - a. Choose *View – Options* and, in the Options dialog box that appears, select the columns that you want to display and deselect the columns that you want to hide.
 - b. Choose *View – Properties* to display or hide property columns.
 - c. Choose *View – Filters* and, in the Filters form that appears, set filters to display specific cellviews.

Only the data that is displayed gets saved.

2. Choose *File – Save Cell Table Data*.

The Save Cell Table Data dialog box appears.



3. In the dialog box, type the name of the file in which to save the data or click *Browse* and use the file browser to find a directory or file.

The default file name is `HE_libName_cellName_configName.csv` and the default location is the current working directory.

The *Cell Bindings* table data is saved to the file you specify. The file contains a line for each row of the table, with items separated by commas. The first line in the file lists the column headers.

For example:

```
Library,Cell,View Found,View to Use  
mixSigLib,cap,behavioral,null  
mixSigLib,and2,layout,behavioral  
mixSigLib,counter,schematic,schematic
```

Virtuoso Hierarchy Editor User Guide

Design Components in Hierarchy Editor

```
mixSigLib, ccadc, schematic, null  
...
```

Related Topics

[Rules Definition at the Cell Level](#)

Virtuoso Hierarchy Editor User Guide

Design Components in Hierarchy Editor

Setting Simulation-Control Properties

The Hierarchy Editor lets you set simulation-control properties that are used by Cadence applications such as the AMS simulator in your configuration.

You can set those properties that have been registered by a Cadence application in the property dictionary. The property dictionary is a central repository of property definitions; it comprises application-specific property dictionary files located in the `your_install_dir/share/cdssetup/registry/props/` directory as well as user property dictionary files, if any. The Hierarchy Editor, by default, lets you display and set all the properties that are in the property dictionary.

You can also set properties that are not in the property dictionary by creating new property columns for them. These are typically properties that you use frequently with an application but that have not been registered in the property dictionary by the application. The Hierarchy Editor saves information about the new property columns in the configuration view in a file named `prop.cfg`.

You can set properties globally for an entire configuration or on cells, instances, or occurrences. Properties are also inherited by the hierarchy below the object on which they are set, unless the property definition specifies that it cannot be inherited.

The Hierarchy Editor stores the values of all the properties that you set in the `prop.cfg` file in the configuration view of your design. The `prop.cfg` file is read by other applications, such as simulators, that use the properties.

Related Topics

[Displaying Properties in Hierarchy Editor](#)

[Virtuoso Hierarchy Editor Editing Form](#)

Displaying Properties in Hierarchy Editor

The Hierarchy Editor displays properties in property columns. These columns are not displayed by default.

To display property columns,

- ➔ Choose *View – Properties*.

The Hierarchy Editor displays one column for each property that is in the property dictionary. It also displays any property columns that you added in a previous session for properties that are not in the property dictionary.

Property columns are displayed in both the table view and the tree view. Properties that are set directly on an object are displayed in blue (the default color for user-defined values); inherited properties are displayed in black.

Property Columns Displayed in Tree View

Instance	View To Use	Inherited View List	sim_mode	speed
(tdmsDemoLib ccadc ...)				
AND2_0 (tdmsDe...		spectreS cdsSpice ...		
B2C_0 (tdmsDemo...		spectreS cdsSpice ...		
CMP_0 (tdmsDem...	schematic	spectreS cdsSpice ...	ms	
Q0 (demoDevi...		spectreS cdsSpice ...	ms	
Q1 (demoDevi...		spectreS cdsSpice ...	ms	
Q10 (demoDe...		spectreS cdsSpice ...	ms	
Q101 (demoD...		spectreS cdsSpice ...	ms	
Q11 (demoDe...		spectreS cdsSpice ...	ms	

Property columns

← sim_mode set on instance

← Inherited sim_mode property

The *View – Properties* command is not available if the `your_install_dir/share/cdssetup/registry/props/` directory or the `your_install_dir/share/cdssetup/ams` directory does not exist in your Cadence tools installation.

You can edit any of the property columns to set properties on your design configuration.

To display property columns by default whenever the Hierarchy Editor is started,

1. Choose *View – Properties*.
2. Choose *File – Save Defaults*.
3. In the Save Defaults dialog box, specify the directory in which to save your default settings, then click *OK*.

Virtuoso Hierarchy Editor User Guide

Setting Simulation-Control Properties

To hide all property columns,

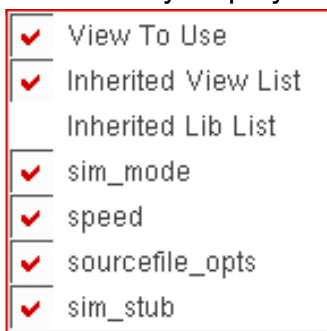
- ➔ Choose *View – Properties*.

Property columns are hidden.

To hide or display individual property columns,

1. Right-click the property column heading.

A pop-up menu, listing all the columns that can be displayed, appears. The columns that are currently displayed have a check-mark next to them.



2. Select the property columns that you want to display and deselect the ones that you want to hide.

The tree view or table view display is updated to reflect your choices. These settings remain in effect for the session only and are not saved in the `hed.env` file when you save your defaults.

To sort the cells in the *Cell Bindings* or *Instance Bindings* table by property,

- ➔ Click the heading of the property column.

To display a description of the property,

- ➔ Place your cursor over the property column heading.

A description of the property is displayed. If the property column has a drop-down list of value choices, you can also place your cursor over each value to see its description. A description of the property and its values is displayed only if it was provided as part of the property definition.

Related Topics

[Properties Setting on the Design](#)

[Adding Property Columns](#)

Virtuoso Hierarchy Editor User Guide

Setting Simulation-Control Properties

Exiting the Hierarchy Editor

Resizing Columns

Properties Setting on the Design

You can set properties on your design globally so that they apply to your entire design configuration, or on cells, instances, or occurrences.

A global property applies to the entire configuration, a cell property applies to all the instances of the cell, an instance property applies to a single instantiation of a cell, and an occurrence property applies to a specific path.

Properties are also inherited by the hierarchy below the object on which they are set, unless a property definition specifies that it cannot be inherited.

Since a property can be set on an object at multiple levels, a certain order of precedence determines the property's value. The Hierarchy Editor uses the following precedence (listed from highest to lowest):

- Occurrence property
- Instance property
- Cell property
- Inherited property
- Global property

Related Topics

[Setting Properties on a Global Basis](#)

[Setting Properties on a Cell](#)

[Setting Properties on an Instance](#)

[Setting Properties on an Occurrence](#)

[Removing Properties](#)

[Adding Property Columns](#)

Setting Properties on a Global Basis

You can set properties on a global basis by setting them on the root cellview of the configuration. Global properties become the default for the configuration and apply to every level of the hierarchy. Global properties can be overridden at the cell, instance, and occurrence levels.

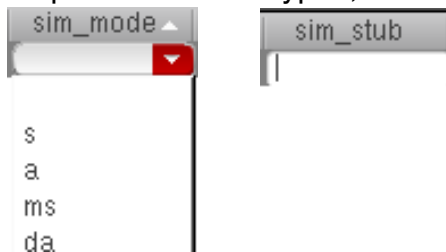
To set a property on a global basis,

1. In the tree view or the table view of the Hierarchy Editor, choose *View – Properties*.

Property columns are displayed for all the properties that are defined in the property dictionary or in the `prop.cfg` file.

2. In the root cellview row (the root cellview is identified by a green pyramid icon in the Information column), click the column of the property you want to set and either type in the value or select it from the drop-down list.

Properties of type `enum` have a drop-down list of values from which you select a value. Properties of other types, such as integer or string, have an editable field. For example:



The property value that you set is displayed in the property column of all objects in the hierarchy, unless it is overridden by other values set at the cell, instance, or occurrence level. The value is displayed in blue (the default color for user-defined values) for the root cellview and in black for all other objects.

If a property definition specifies that it cannot be inherited, then it is not inherited by the hierarchy below it even if it is set at the global level.

Related Topics

[Properties Setting on the Design](#)

[Setting Properties on a Cell](#)

[Setting Properties on an Instance](#)

[Setting Properties on an Occurrence](#)

Removing Properties

Setting Properties on a Cell

A cell-level property applies to every instantiation of the cell. It is also inherited by all objects below the cell in the hierarchy, unless its definition specifies that it cannot be inherited.

Cell-level properties override global properties and can be overridden at the instance and occurrence levels.

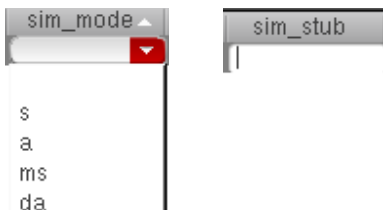
To set a property on a cell

1. Choose *View – Parts Table* to display the table view of the Hierarchy Editor.
2. Choose *View – Properties*.

Property columns are displayed for all the properties that are defined in the property dictionary or in the `prop.cfg` file.

3. In the cell row, click the column of the property you want to set and either type in the value or select it from the drop-down list.

Properties of type `enum` have a drop-down list of values from which you select a value. Properties of other types, such as integer or string, have an editable field. For example:



The property value appears in the property column of the cell in the Cell Bindings section. Because the property is also inherited by the hierarchy below the cell, the property value also appears in the property columns of all the cells that are under that cell. A property does not get inherited by the hierarchy below the cell if the property definition specifies that it cannot be inherited.

Viewing Instantiations

To see a list of the instantiations to which the property applies,

1. In the *Cell Bindings* section, right-click the cell.
2. From the pop-up menu, select *Explain*.

The Explain dialog box appears. The Instantiations section lists the instantiations of the cell.

To see the hierarchy below the cell to which the property also applies,

1. Choose *View – Tree* to display the tree view of the configuration.
2. Double-click the cell or click its expansion icon.

Related Topics

[Properties Setting on the Design](#)

[Setting Properties on a Global Basis](#)

[Setting Properties on an Instance](#)

[Setting Properties on an Occurrence](#)

[Removing Properties](#)

Setting Properties on an Instance

Instance properties apply to only one instantiation of a cell (unlike cell properties that apply to all the instantiations of a cell). However, that instance properties can apply to multiple objects—if the cell that contains the instance is used in multiple places in the design, the property applies to the instance in all those locations.

Instance properties are also inherited by all objects in the hierarchy below the instance, unless a property's definition specifies that it cannot be inherited.

Instance-level properties override properties set at the global or cell level and can be overridden at the occurrence level.

Setting Instance Properties from the Tree View

To set a property on an instance from the tree view of the Hierarchy Editor,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. If the *Occurrence Mode* button on the toolbar is selected, deselect it.

The tree view now displays *Target: Instance*.

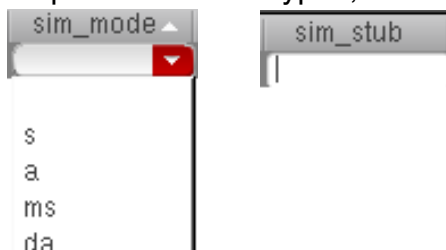
If an instance has already been identified as an occurrence, the tree view displays *Target: Occurrence* when it is selected, and any property you set on it gets added to the occurrence, not the instance.

3. Choose *View – Properties*.

Property columns are displayed for all the properties that are defined in the property dictionary or in the `prop.cfg` file.

4. In the *Tree View* section, click the property column of the instance on which you want to set the property and either type in the value or select it from the drop-down list.

Properties of type `enum` have a drop-down list of values from which you select a value. Properties of other types, such as integer or string, have an editable field. For example:



Setting Instance Properties from the Table View

To set a property on an instance from the table view,

1. Choose *View – Parts Table* and *View – Instance Table* to display the table view of the Hierarchy Editor, if it is not already displayed.
2. Choose *View – Properties*.


Property columns are displayed for all the properties that are defined in the property dictionary or in the `prop.cfg` file.

3. In the *Cell Bindings* section, select the cell that contains the instance on which you want to set the property.

The *Instance Bindings* section displays all the instances the cell contains.

4. In the *Instance Bindings* section, click in the property column of the instance on which you want to set the property and either type in the value or select it from the drop-down list.

Properties of type `enum` have a drop-down list of values from which you select a value. Properties of other types, such as integer or string, have an editable field.

You cannot add an instance property to an object that has already been defined as an occurrence (identified by a  icon). To do so, you must remove the occurrence binding first.

To see which objects the property applies to,

1. In the *Cell Bindings* section in the table view, right-click the cell that contains the instance on which you set the property.
2. From the pop-up menu, select *Explain*.

The Explain dialog box appears. The Instantiations section lists the instantiations of the cell. The property gets applied to the instance in all these instantiations of its parent cell.

To see the hierarchy below the instance to which the property also applies,

1. Choose *View – Tree* to display the tree view of the configuration.
2. Double-click the instance or click its expansion icon.

Related Topics

[Properties Setting on the Design](#)

Virtuoso Hierarchy Editor User Guide

Setting Simulation-Control Properties

[Setting Properties on a Cell](#)

[Setting Properties on a Global Basis](#)

[Setting Properties on an Occurrence](#)

[Removing Properties](#)

Setting Properties on an Occurrence

You can also set properties on an occurrence. An occurrence property is set on a specific path. It is also inherited by all objects in the hierarchy below the occurrence, unless the property definition specifies that it cannot be inherited.

An occurrence property has the highest precedence; it cannot be overridden by a property set at any other level of the hierarchy.

You can only add an occurrence property in the tree view of the Hierarchy Editor; you cannot add it in the table view.

To set an occurrence property,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. Turn on the occurrence editing mode by clicking the *Set bindings...on Occurrence* button in the toolbar:

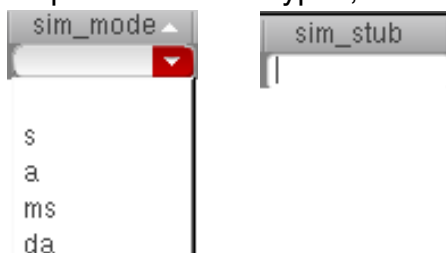
The tree view displays *Target: Occurrence*.


3. Choose *View – Properties*, if necessary.

Property columns are displayed for all the properties that are defined in the property dictionary or in the `prop.cfg` file.

4. In the *Tree View* section, click in the property column of the occurrence on which you want to set the property, and either type in the value or select it from the drop-down list.

Properties of type `enum` have a drop-down list of values from which you select a value. Properties of other types, such as integer or string, have an editable field. For example:



The property value appears in the property column with the  icon (occurrence icon) next to it. The path is now identified uniquely as an occurrence; if any other attribute is set on it, such as a stop point, it is set on the occurrence, not the instance.

Related Topics

[Properties Setting on the Design](#)

[Setting Properties on a Cell](#)

[Setting Properties on a Global Basis](#)

[Setting Properties on an Instance](#)

[Removing Properties](#)

Removing Properties

You can remove a property in two ways:

- By removing it from the object on which it is set
- By removing it from the entire design configuration

Removing Properties from Specific Objects

You can remove a property from a specific object on which it is set—that is, from a cell, instance, or occurrence.

When you remove a property from an object, it is also removed from all the objects in the hierarchy below it that inherited the property.

To remove a property from a specific object,

1. If you are removing a property from a cell, display the table view of the Hierarchy Editor; if you are removing it from an instance, display either the table view or the tree view; if you are removing it from an occurrence, display the tree view and turn on the occurrence editing mode.
2. In the object row, click in the property column of the property that you want to remove.
3. Either delete the text in the field, or, if a drop-down list appears, set the value to the empty space at the top of the list.

Note: You can only remove a property from objects on which it is set directly; you cannot remove it from objects that have inherited the property.

Removing Properties from the Entire Design Configuration

To remove a property from all objects on which it is set, that is to remove all references to it from the design configuration, you delete its property column.

You can delete only those property columns that you have added with the *Edit – Add Property Column* command. You cannot delete property columns of properties that are defined in the property dictionary.

Note: When you remove a property column, you are removing that property from all objects on which it is set. You cannot be able to undo the delete command. Therefore, remove a column only if you are sure that you want to remove the property entirely from your design.

Virtuoso Hierarchy Editor User Guide

Setting Simulation-Control Properties

To delete a property column,

1. Choose *Edit – Remove Property Column*.

The Remove a Property column dialog box appears.



2. In the dialog box, select the property column that you want to delete.

The list only displays those property columns that you have added with the *Edit – Add Property Column* command.

3. Click *OK*.

Related Topics

[Properties Setting on the Design](#)

[Setting Properties on a Cell](#)

[Setting Properties on a Global Basis](#)

[Setting Properties on an Instance](#)

Adding Property Columns

The Hierarchy Editor, by default, lets you display and set all the simulation-control properties that are in the property dictionary. If you want to set a property that is not in the property dictionary, you can do so by creating a property column for it. You typically add properties that you use frequently with an application.

You should only add simulation-control properties that are understood by a Cadence application.

When you create a new property column, set the property on objects in your design, and then save the configuration, the new property column definition is added to the `prop.cfg` file in the configuration view. If a `prop.cfg` file does not exist, it is created.

The `prop.cfg` file is a property file created by the Hierarchy Editor in the configuration view. It stores the values of all properties that you set with the Hierarchy Editor. It also stores information about any property columns that you add. The `prop.cfg` file is read by applications that use the properties.

Note: Do not edit the `prop.cfg` file manually.

The next time that you display properties with the *View – Properties* command, the property column that you added is always displayed.

To add a property column,

1. Choose *Edit – Add Property Column*.

The Add a Property Column dialog box appears.



The *Edit – Add Property Column* command is only available if the *View – Properties* command is selected.

2. In the *Property Name* field, specify the name of the property for which you want to add a property column.

The name must match the property name that the application recognizes.

Virtuoso Hierarchy Editor User Guide

Setting Simulation-Control Properties

3. In the *Property Type* field, select the property type from the pulldown menu. The following choices are available: *String*, *Int*, *Double*.

The Hierarchy Editor uses the property type you specify to do type-checking when the property is set.

4. Click *Apply* to display the new property column or *OK* to display the new property column and close the dialog box.

The new property column is displayed.

5. Set the new property on objects in your design configuration.
6. Choose *File – Save* to save your configuration.

If you create a new property column for a property but do not set the property on any objects in your configuration, the new property column definition is not saved when you save the configuration. For the new definition to be saved, the property must be set on at least one object in your configuration.

Related Topics

[Properties Setting on the Design](#)

[Displaying Properties in Hierarchy Editor](#)

[Removing Properties](#)

Hierarchy Editor Plug-Ins

Plug-ins are applications that are added to the Hierarchy Editor and used from the Hierarchy Editor user interface. Some Cadence products have been added to the Hierarchy Editor as plug-ins.

Note: You cannot add any other Cadence products, or your own applications, as plug-ins.

If at least one plug-in has been added to the Hierarchy Editor, the Hierarchy Editor menu bar displays the *Plugins* menu. If the *Plugins* menu is not present, no plug-ins are available with the Hierarchy Editor.

Related Topics

[Loading Plug-In Applications](#)

[Checking Imported Packages](#)

Loading Plug-In Applications

The *Plugins* menu contains all the applications that are available from the Hierarchy Editor. To use any application listed in the menu, you must first load it. The Hierarchy Editor does not load the plug-ins by default.

When you load a plug-in application, it appears in the Hierarchy Editor, typically as a menu on the menu bar or an icon on the tool bar. You can load a plug-in either from the command line when you start the Hierarchy Editor or from the Hierarchy Editor user interface.

Only those Cadence applications that are predefined as Hierarchy Editor plug-ins are displayed in the *Plugins* menu. You cannot add any other Cadence applications, or your own menu items, to the menu.

Loading a Plug-In from the Hierarchy Editor User Interface

To load a plug-in application from the Hierarchy Editor user interface,

1. Start the Hierarchy Editor.

The *Plugins* sub-menu appears in the *Launch* menu. The menu lists the applications that you can load. A check mark next to a menu item indicates the application is already loaded.

The *Plugins* sub-menu appears only if plug-ins have been added to the Hierarchy Editor.

2. Choose *Launch – Plugins* and select a plugin that you want to load, for example, *AMS HDL Settings*.
3. The application appears in the Hierarchy Editor as specified by the plug-in. For instance, it could appear as an additional menu in the Hierarchy Editor menu bar or an icon in the Hierarchy Editor tool bar.
4. (Optional) If you want the Hierarchy Editor to automatically load the plug-in application every time it starts, save the Hierarchy Editor defaults with the *File – Save Defaults* command.

Loading a Plug-in from the Command Line

To load a plug-in application from the command line,

1. Start the Hierarchy Editor with the following command:

```
cdsHierEditor -plugin pluginName [pluginOptions]
```


Virtuoso Hierarchy Editor User Guide

Hierarchy Editor Plug-Ins

To load more than one plug-in, use the following format:

```
cdsHierEditor -plugin pluginName [pluginOptions] -plugin pluginName  
[pluginOptions] ...
```

`-plugin` must be the last argument specified with the `cdsHierEditor` command. It can be followed only by other `-plugin` arguments.

When the Hierarchy Editor starts, the plug-in application appears in the Hierarchy Editor graphical user interface, typically as a menu in the menu bar or an icon in the tool bar.

2. (Optional) If you want the Hierarchy Editor to automatically load the plug-in application every time it starts, save the Hierarchy Editor defaults with the *File – Save Defaults* command.

Related Topics

[Checking Imported Packages](#)

[Removing Plug-In Applications](#)

Checking Imported Packages

The *Package Importing Checker* functionality allows you to view a list of imported packages, comprising VHDL and SystemVerilog (SV) modules and their dependency relationships. In addition, it enables you to find errors and rectify them during simulation before netlisting a design. This helps in reducing the errors that are generated during netlisting and saves time.

To view the list of imported packages using the built-in AMS UNL plugin available in HED, perform the following steps:

1. To enable AMS UNL plugin, choose *Launch – Plugins – AMS HDL Settings* from the Virtuoso Hierarchy Editor window.
2. Once the plugin is enabled, the *AMS UNL* menu is added to the HED menu. To view the list of imported packages, choose *AMS UNL – Package Importing Checker*. The Package Checking Viewer form is displayed.

Related Topics

[Open Form](#)

Pin Checker

Pin Checker is a tool that checks the connections between an instance and its master in the config design. A config design can consist of both text and non-text cellviews.

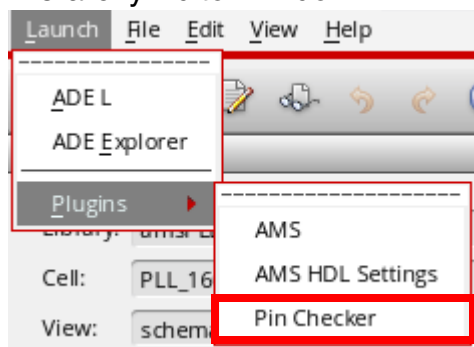
Pin Checker helps you to find the mismatch in the connectivity and allows you to fix the issues before netlisting. This, in turn, helps in reducing the errors that are generated during netlisting and saves time.

Pin Checker supports the hierarchy of text and non-text views. The support for text cellview is available for VerilogD, VerilogAMS, SystemVerilog, VHDL, and VHDLAMS.

It performs the following checks:


- If the parent containing the instance is text, then it checks the connectivity of the instance and its switch master. It compares the number and size of terminals in the switch master with the instance.
- If the parent containing the instance is schematic, then it performs the following checks:
 - Checks the connectivity of the instance and its place master. It compares the number of terminals in the place master with the instance.
 - Checks the connectivity of instance's place master and switch master. It compares the number and direction of terminals between the switch master and the place master.

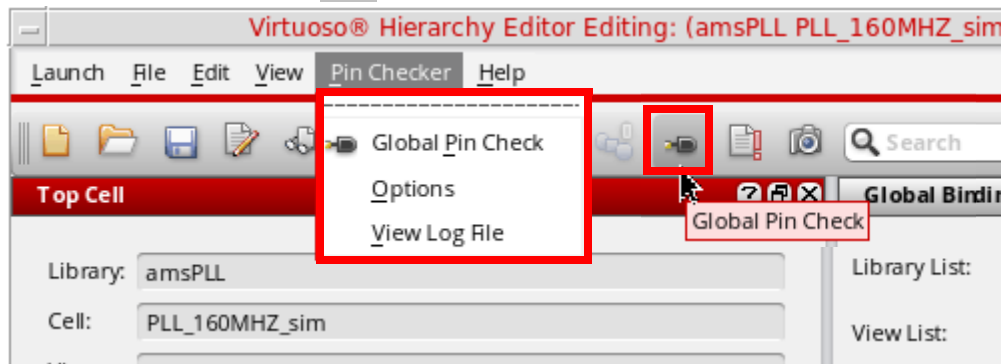
To enable Pin Checker, select *Launch — Plugins — Pin Checker* from the Virtuoso Hierarchy Editor window.



Virtuoso Hierarchy Editor User Guide

Hierarchy Editor Plug-Ins

Once the plugin is enabled, the *Pin Checker* menu is added to the HED menu items. In addition, the *Global Pin Check*  button is also displayed on the HED toolbar.



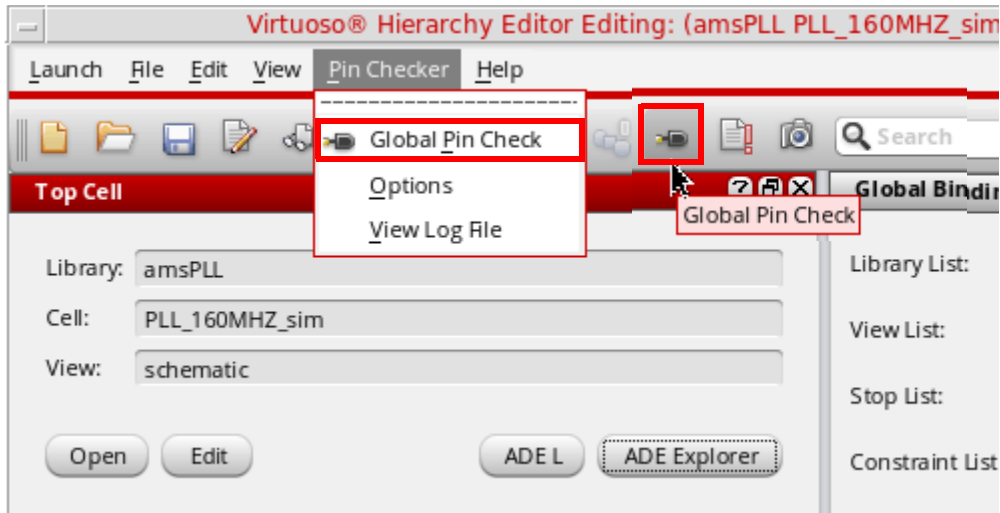
Related Topics

[Pin Checker Options Form](#)

Performing a Global Pin Check

You can perform hierarchical pin check on all the instances in a design. To do this, you need to perform one of the following steps:

- Select *Global Pin Check* from the *Pin Checker* menu.
- Click the *Global Pin Check* button on the HED toolbar.



After the pin checking operation on a design is successfully completed without any error or warning, a successful pin check message is displayed in CIW.

If there is any error or warning in the design, the unsuccessful pin check message is displayed in CIW and the pin check log file is displayed.

Related Topics

Log Files Creation

Performing a Single Instance Pin Check

To perform pin checking on a single instance, in the *Tree View* of the Virtuoso Hierarchy Editor window,

- ➔ Right-click the instance and select *Pin Check Instance* from the context-sensitive menu.

After pin checking is successfully completed without any error or warning on the selected instance, the successful pin check message is displayed in CIW.

If there is any error or warning in the design, the unsuccessful pin check message is displayed in CIW and the Pin Check log file is displayed.

Additionally, you can perform pin checking on a particular instance of the selected cell. To do this, you need to perform the following steps:

1. Open the *Instance Bindings* section in the *Tree View* by selecting *View – Instance Table* from the Virtuoso Hierarchy Editor window. The *Instance Bindings* section is displayed in the *Table View* that lists all the instances of the cell, which is selected in the *Cell Bindings* section.
2. To perform pin checking on a single instance of the selected cell, right-click the instance in the *Instance Bindings* section and select *Pin Check Instance* from the context-sensitive menu.

Related Topics

[Log Files Creation](#)

Performing Multiple Instances Pin Check

You can select multiple instances for pin checking. To do this, you need to perform the following steps:

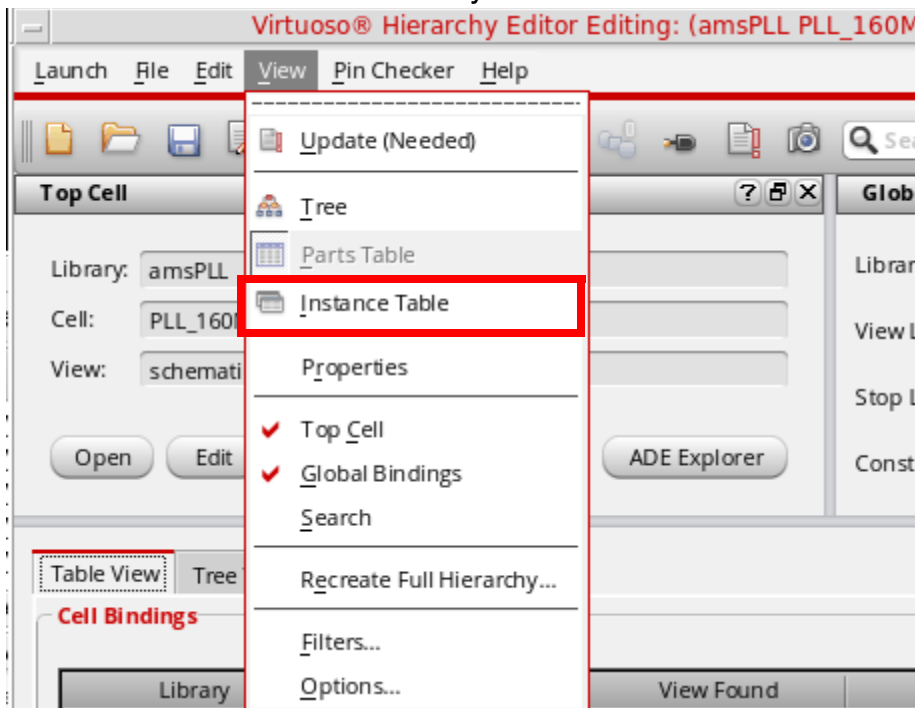
1. In the *Tree View* of the Virtuoso Hierarchy Editor window, select multiple instances by using the **Ctrl** or **Shift** key with the left mouse button.
2. Right-click the selected instances and choose *Pin Check Multiple Selections* from the context-sensitive menu.

After pin checking is successfully completed without any error or warning on multiple instances, the successful pin check message is displayed in CIW.

If there is any error or warning in the design, the unsuccessful pin check message is displayed in CIW and the Pin Check log file is displayed.

Additionally, you can perform pin checking on multiple instances of the selected cell. To do this, you need to perform the following steps:

1. Open the *Instance Bindings* section in the *Tree View* by selecting *View – Instance Table* from the Virtuoso Hierarchy Editor window.



2. The *Instance Bindings* section is displayed in the *Table View* that lists all the instances of the cell, which is selected in the *Cell Bindings* section. To perform pin checking on

multiple instances of the selected cell, select multiple instances from the *Instance Bindings* section by using the `Ctrl` or `Shift` key with the left mouse button.

3. Right-click the selected instances and select *Pin Check Multiple Instances* from the context-sensitive menu.

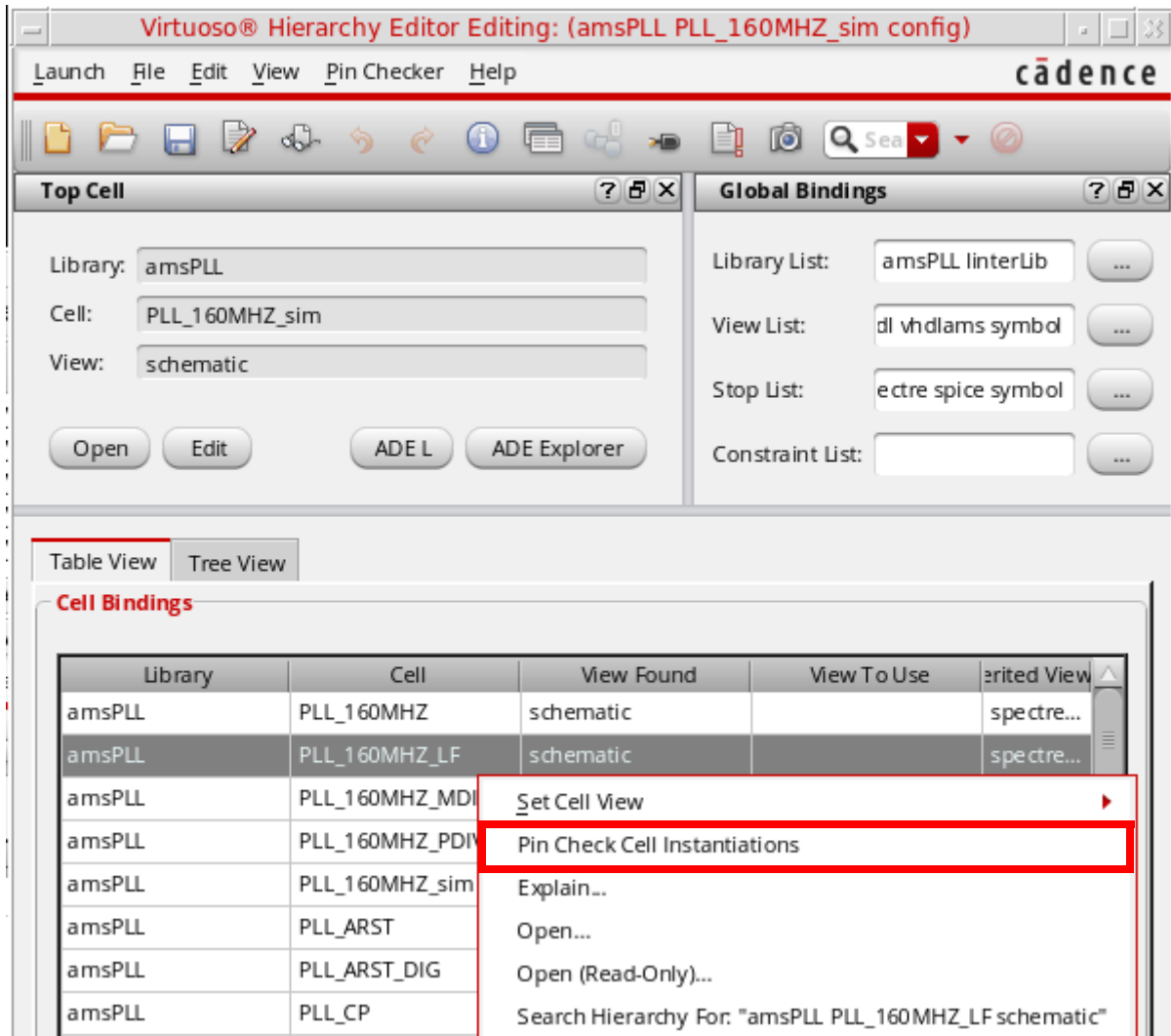
Related Topics

[Log Files Creation](#)

Performing a Cell Instantiation Pin Check

You can perform pin checking on a particular cell at all the places where it is instantiated in a design. To do this, in the *Table view* of the Virtuoso Hierarchy Editor window,

- ➔ Right-click the *Cell Bindings* row and select *Pin Check Cell Instantiations* from the context-sensitive menu.



After cell instantiation is successfully completed without any error or warning on all the instances, the successful pin check message is displayed in CIW.

If there is any error or warning in the design, the unsuccessful pin check message is displayed in CIW and the Pin Check log file is displayed.

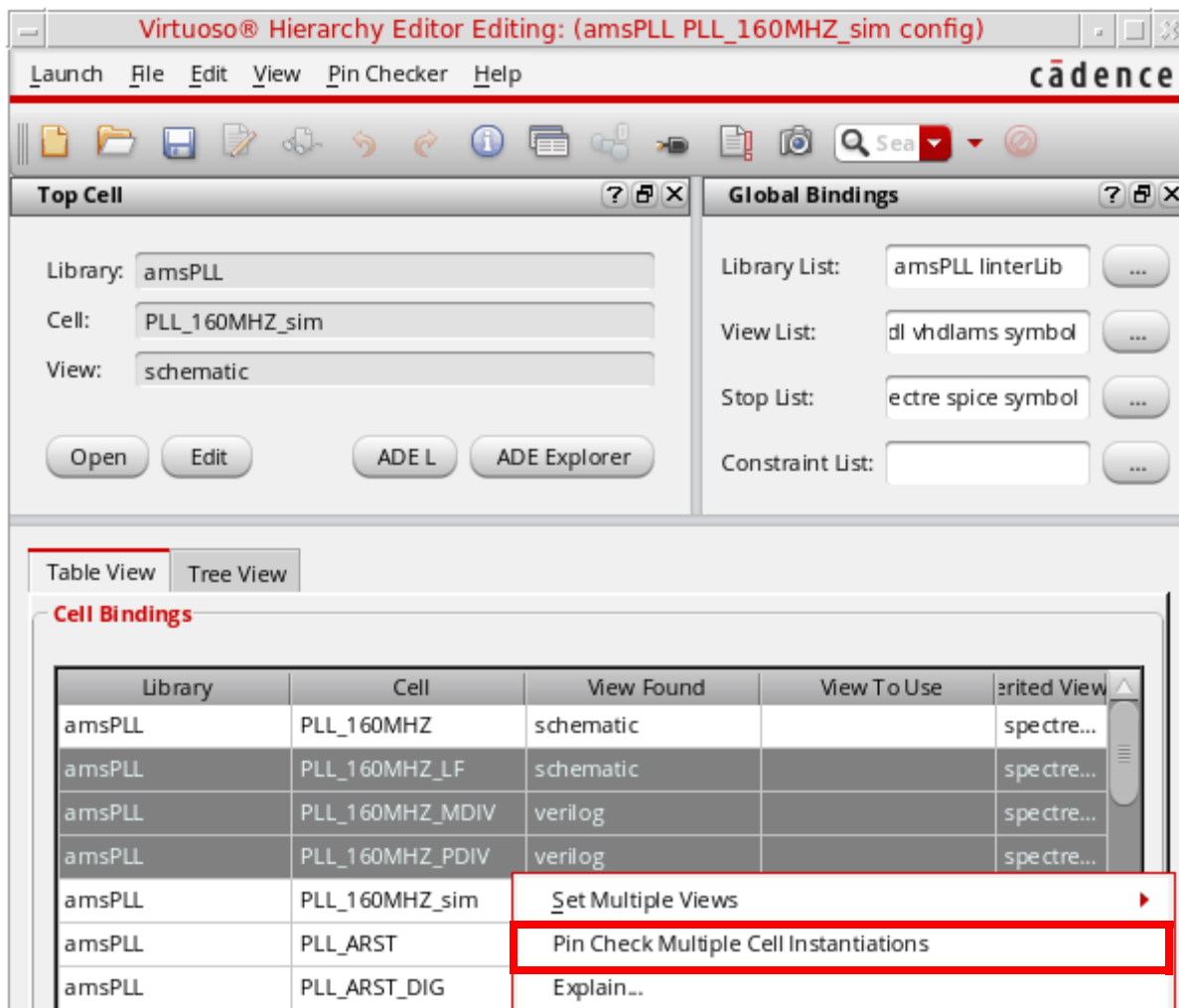
Virtuoso Hierarchy Editor User Guide

Hierarchy Editor Plug-Ins

Multiple Cell Instantiations Pin Check

You can perform pin checking on multiple cells at all the places where they are instantiated in a design. To do this, you need to perform the following steps:

1. In the *Table view* of the Virtuoso Hierarchy Editor window, select multiple cells from the *Cell Bindings* section by using the **Ctrl** or **Shift** key with the left mouse button.
2. Right-click the selected cells and select *Pin Check Multiple Cell Instantiations* from the context-sensitive menu



After Pin Check for instantiations of multiple cells is successfully completed without any error or warning, the successful pin check message is displayed in CIW.

If there is any error or warning in the design, the unsuccessful pin check message is displayed in CIW and the Pin Check log file is displayed.

Virtuoso Hierarchy Editor User Guide

Hierarchy Editor Plug-Ins

Related Topics

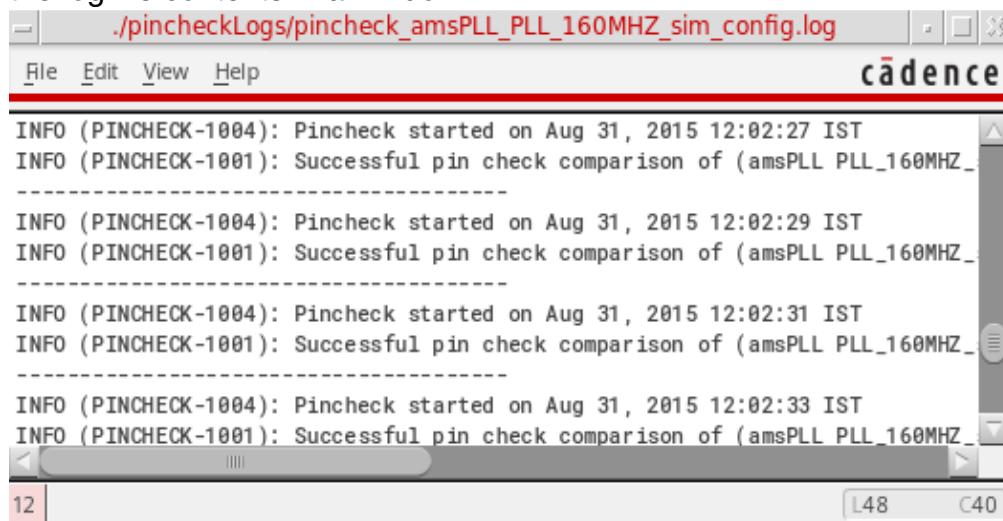
[Log Files Creation](#)

Log Files Creation

A log file with the name `pincheck_<library name>_<cell name>_<view name>.log` is created in the current working directory, by default. Here, `<library_name>`, `cell_name`, `view_name` are the library, cell, and view names of the current config window where pin checking has been performed. This log file contains the error and warning messages if the pin checker finds any issues during the pin checking operation. For successive pin checking operations, error and warning messages are appended in the log file. A log file window is also displayed showing the content of the log file.

For example, if you are working on a design where the library name is `amsPLL`, the cell name is `PLL_160MHZ_sim`, and the view name is `config`. Then, the Pin Check log file is created with the name, `pincheck_amsPLL_PLL_160MHZ_sim_config.log`.

To view the log file in a window, you need to select *Pin Checker — View Log File*. It shows the log file contents in a window.



Related Topics

[pcLogFileOpenMode](#)

[pcLogFilePath](#)

Removing Plug-In Applications

To remove a plug-in application from the Hierarchy Editor,

- ➡ From the *Plugins* menu, choose the application that you want to remove.

The application menu or toolbar icon disappears from the Hierarchy Editor. Also, in the *Plugins* menu, the application name no longer has a check mark next to it, which indicates that the plug-in is not loaded.

You can load the plug-in application again by selecting it from the *Plugins* menu.

Related Topics

[Loading Plug-In Applications](#)

Virtuoso Hierarchy Editor User Guide

Hierarchy Editor Plug-Ins

Hierarchy Editor Environment Variables

This section provides information on the names, descriptions, and graphical user interface equivalents for the hierarchy editor environment variables.

The local `hed.env` file is typically created with the *File – Save Defaults* command and must be in a directory that is listed in your `setup.loc` file (for example, your home directory or current working directory). The local `hed.env` file contains only those settings that are different from the registration `hed.env` file.

The `hed.env` file is read by the Hierarchy Editor when it is started. If you have multiple `hed.env` files, the Hierarchy Editor determines which value to use for each variable based on the search order defined in the `setup.loc` file. For more information about the `setup.loc` file, see, [Cadence Setup Search File: setup.loc](#).

Some of the form and display settings, typically the general user interface settings, are saved when you use the *File – Save Defaults* command. For example, the options that you select in the Options form or in the Filters form are saved when you use the *File – Save Defaults* command. Other settings, such as the list of views in the Edit Constants form, can only be changed by editing the `hed.env` file manually.

You can use a semicolon to comment out a line in your `hed.env` file. The comment character must be the first character in the line. For example:

```
; My defaults
```

```
hed.constants viewChoices string "schematic symbol verilog verilogAMS"
```

List of Hierarchy Editor Environment Variables

[setAmsSimulator](#)

[bindingError](#)

[defaultBinding](#)

[highlight](#)

Virtuoso Hierarchy Editor User Guide

Hierarchy Editor Environment Variables

notInUse

rowSelection

userBinding

maskLayoutStopLimit

viewChoices

maskLayoutStopLimit

showConstInViewList

filter

autoGenPcdbFiles

useNCCompilers

outputDir

spaceIndent

minTraversal

subConfigs

diffTool

pcLogFileOpenMode

pcLogFilePath

setAmsSimulator

```
hed.ade setAmsSimulator boolean { t | nil }
```

Description

Disables the default `ams` simulator setting.

After setting this variable, simulator `ams` or `spectre` is determined by the `asimenv.startup simulator` environment variable when a new ADE Explorer view is created.

If a simulator value is specified through the simulator constant in the Edit Constants form for Virtuoso Hierarchy Editor, it takes precedence over the value set through the `asimenv.startup simulator` environment variable.

The default value is `t`.

GUI Equivalent

None

Examples

```
envGetVal("hed.ade" "setAmsSimulator")  
envSetVal("hed.ade" "setAmsSimulator" 'boolean nil)
```

Related Topics

[asimenv.startup simulator](#)

[Setting Up the Default Simulator for a Configuration](#)

[Edit Constants Form](#)

[List of Hierarchy Editor Environment Variables](#)

showInfo

```
hed.cellTable showInfo boolean { t | nil }
```

Description

Specifies whether to display the icons associated with settings, such as stop points and file bindings, in the information column of the Cell table.

The default value is `t`.

GUI Equivalent

None

Examples

```
envGetVal("hed.cellTable" "showInfo")  
envSetVal("hed.cellTable" "showInfo" 'boolean nil)
```

Related Topics

[showInfo](#) (For the Instance table)

bindingError

```
hed.colorChooser bindingError string "color"
```

Description

Defines the color used to display data that has a binding error.

Specify the color by name or by RGB value. The name can be any name from the list of SVG color keyword names provided by the [World Wide Web Consortium](https://www.w3.org/TR/css3-color/#svg-color).

The default color is `red`.

GUI Equivalent

None

Examples

```
envGetVal("hed.colorChooser" "bindingError")  
envSetVal("hed.colorChooser" "bindingError" 'string "yellow")
```

Related Topics

[List of Hierarchy Editor Environment Variables](#)

[defaultBinding](#)

[highlight](#)

[notInUse](#)

[rowSelection](#)

[userBinding](#)

defaultBinding

```
hed.colorChooser defaultBinding string "color"
```

Description

Defines the color used to indicate the default binding based on the binding rules.

Specify the color by name or by RGB value. The name can be any name from the list of SVG color keyword names provided by the [World Wide Web Consortium](https://www.w3.org/TR/css3-color/#svg-color).

The default color is `black`.

GUI Equivalent

None

Examples

```
envGetVal("hed.colorChooser" "defaultBinding")
envSetVal("hed.colorChooser" "defaultBinding" 'string "red")
```

Related Topics

[List of Hierarchy Editor Environment Variables](#)

[viewChoices](#)

[highlight](#)

[notInUse](#)

[rowSelection](#)

[userBinding](#)

highlight

```
hed.colorChooser highlight string "color"
```

Description

Defines the color used to highlight an object that an external application wants to find. For example, Virtuoso schematic editor users can add an instance probe and see it in the Hierarchy Editor.

Specify the color by name or by RGB value. The name can be any name from the list of SVG color keyword names provided by the [World Wide Web Consortium](#).

The default highlight color is khaki.

GUI Equivalent

None

Examples

```
envGetVal("hed.colorChooser" "highlight")
envSetVal("hed.colorChooser" "highlight" 'string "green")
```

Related Topics

[List of Hierarchy Editor Environment Variables](#)

[defaultBinding](#)

[viewChoices](#)

[notInUse](#)

[rowSelection](#)

[userBinding](#)

notInUse

```
hed.colorChooser notInUse string "color"
```

Description

Defines the color used to indicate objects that are not in use.

Specify the color by name or by RGB value. The name can be any name from the list of SVG color keyword names provided by the [World Wide Web Consortium](https://www.w3.org/TR/css3-color/#svg-color).

The default color is `darkOrange`.

GUI Equivalent

None

Examples

```
envGetVal("hed.colorChooser" "notInUse")  
envSetVal("hed.colorChooser" "notInUse" 'string "green")
```

Related Topics

[List of Hierarchy Editor Environment Variables](#)

[defaultBinding](#)

[highlight](#)

[rowSelection](#)

[userBinding](#)

rowSelection

```
hed.colorChooser rowSelection string "color"
```

Description

Defines the color used to indicate a row selection in the table and the tree. Cadence recommends that you do not change the default color, which is set according to Cadence style.

GUI Equivalent

None

Examples

```
envGetVal("hed.colorChooser" "rowSelection")
envSetVal("hed.colorChooser" "rowSelection" 'string "green")
```

Related Topics

[List of Hierarchy Editor Environment Variables](#)

[defaultBinding](#)

[highlight](#)

[notInUse](#)

[userBinding](#)

userBinding

```
hed.colorChooser userBinding string "color"
```

Description

Defines the color used to indicate the bindings that you have changed.

Specify the color by name or by RGB value. The name can be any name from the list of SVG color keyword names provided by the [World Wide Web Consortium](#).

The default color is `blue`.

GUI Equivalent

None

Examples

```
envGetVal("hed.colorChooser" "userBinding")
envSetVal("hed.colorChooser" "userBinding" 'string "green")
```

Related Topics

[List of Hierarchy Editor Environment Variables](#)

[defaultBinding](#)

[highlight](#)

[notInUse](#)

[rowSelection](#)

[filter](#)

maskLayoutStopLimit

```
hed.configCompare maskLayoutStopLimit int integer_number
```

Description

Restricts the opening of maskLayout databases when a config traversal or comparison is performed and maskLayout cellview exceeds the specified limit.

The default value is 100.

GUI Equivalent

None

Examples

```
envGetVal("hed.configCompare" "maskLayoutStopLimit")  
envSetVal("hed.configCompare" "maskLayoutStopLimit" 'int 50)
```

Related Topics

[List of Hierarchy Editor Environment Variables](#)

[defaultBinding](#)

[highlight](#)

[notInUse](#)

[rowSelection](#)

[filter](#)

viewChoices

```
hed.constants viewChoices string "view1 view2 view3 ..."
```

Description

Defines the list of views in the *View Choices* list box in the Edit Constants form.

GUI Equivalent

Command	<i>Edit – Constants</i>
Form Field	<i>Views</i>

Examples

```
envGetVal("hed.constants" "viewChoices")  
envSetVal("hed.constants" "viewChoices" 'string "green")
```

Related Topics

List of Hierarchy Editor Environment Variables

defaultBinding

highlight

notInUse

rowSelection

filter

maskLayoutStopLimit

```
hed.display maskLayoutStopLimit int integer_number
```

Description

Specifies a size limit for maskLayout cellview databases. MaskLayout databases, such as extracted cellview databases, larger than the specified integer value (in MBs) opened and displayed as stopping in Hierarchy Editor.

The default value is 100. If 0 is specified, all maskLayout cellviews are displayed as stopping.

GUI Equivalent

None

Examples

```
envGetVal("hed.display" "maskLayoutStopLimit")  
envSetVal("hed.display" "maskLayoutStopLimit" 'int 50)
```

Related Topics

List of Hierarchy Editor Environment Variables

defaultBinding

highlight

notInUse

rowSelection

filter

showConstInViewList

```
hed.display showConstInViewList boolean { t | nil }
```

Description

Specifies that instead of displaying a constant, the Hierarchy Editor should display the expanded view list that the constant represents.

The default value of this variable is `t`.

GUI Equivalent

None

Examples

```
envGetVal("hed.display" "showConstInViewList")  
envSetVal("hed.display" "showConstInViewList" 'boolean nil)
```

Related Topics

[List of Hierarchy Editor Environment Variables](#)

[defaultBinding](#)

[highlight](#)

[notInUse](#)

[rowSelection](#)

[filter](#)

showInfo

```
hed.instTable showInfo boolean { t | nil }
```

Description

Specifies whether to display the icons associated with settings, such as stop points and file bindings, in the information column of the Instance table.

The default value is `t`.

GUI Equivalent

None

Examples

```
envGetVal("hed.instTable" "showInfo")
envSetVal("hed.instTable" "showInfo" 'boolean nil)
```

Related Topics

[showInfo](#) (For the Cell table)

filter

```
hed.saveAsVerilog filter string "Any Files (*)"
```

Description

Defines the filter that is available in the Save As Verilog form. The default filter is "Any Files (*)". To change this to another filter, use the following format:

```
[optional comment] (space or semicolon-separated expressions)
```

GUI Equivalent

None

Examples

```
envGetVal("hed.saveAsVerilog" "filter")  
envSetVal("hed.saveAsVerilog" "filter" 'string "Verilog Files (*.v)")
```

Related Topics

[List of Hierarchy Editor Environment Variables](#)

[defaultBinding](#)

[highlight](#)

[notInUse](#)

[rowSelection](#)

name

```
hed.template name string "name_of_the_template"
```

Description

Sets the default name of the configuration template in the Use Template form.

The default is " ".

GUI Equivalent

Command	<i>File – New Configuration – Use Template</i>
Form Field	<i>Name</i>

Examples

```
envGetVal("hed.template" "name")  
envSetVal("hed.template" "name" 'string "spectre")
```

Related Topics

[Use Template Form](#)

[New Configuration Form](#)

autoGenPcdbFiles

```
hed.update autoGenPcdbFiles boolean { t | nil }
```

Description

Specifies that the Hierarchy Editor should automatically generate a `pc.db` file from Verilog or VHDL.

GUI Equivalent

None

Examples

```
envGetVal("hed.update" "autoGenPcdbFiles")  
envSetVal("hed.update" "autoGenPcdbFiles" 'boolean nil)
```

Related Topics

[List of Hierarchy Editor Environment Variables](#)

[defaultBinding](#)

[highlight](#)

[notInUse](#)

[rowSelection](#)

[filter](#)

useNCCompilers

```
hed.update useNCCompilers boolean { t | nil }
```

Description

Specifies that the NC compilers (NC-Verilog and NC-VHDL) be used when generating a `pc.db` file from Verilog or VHDL. `useNCCompilers` is used in conjunction with the `autoGenPcdbFiles` variable.

GUI Equivalent

None

Examples

```
envGetVal("hed.update" "useNCCompilers")  
envSetVal("hed.update" "useNCCompilers" 'boolean nil)
```

Related Topics

[List of Hierarchy Editor Environment Variables](#)

[defaultBinding](#)

[highlight](#)

[notInUse](#)

[rowSelection](#)

[filter](#)

Compare Configuration Variables

This section provides information on the names, descriptions, and graphical user interface equivalents for the Compare Configuration environment variables.

You can use these environment variables to control the traversal output displayed while performing the configuration traversal and comparison:

- outputDir
- spaceIndent
- minTraversal
- subConfigs
- diffTool

outputDir

```
hed.configCompare outputDir string "./outputfilepath"
```

Description

Specifies an output directory to save the traversed configuration files. The default output directory is `./configTraversals`.

GUI Equivalent

None

Examples

```
envGetVal("hed.configCompare" "outputDir")  
envSetVal("hed.configCompare" "outputDir" 'string "./configFile")
```

Related Topics

[Compare Configuration Variables](#)

spaceIndent

```
hed.configCompare spaceIndent string "spaces"
```

Description

Specifies the number of spaces to be used for indentation.

The default value is 2 and the value ranges from 1 to 8.

GUI Equivalent

None

Examples

```
envGetVal("hed.configCompare" "spaceIndent")  
envSetVal("hed.configCompare" "spaceIndent" 'string "4")
```

Related Topics

[Compare Configuration Variables](#)

minTraversal

```
hed.configCompare minTraversal boolean { t | nil }
```

Description

Performs the minimal traversal of the configuration hierarchy. When set to `nil`, full traversal is performed.

The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("hed.configCompare" "minTraversal")
envSetVal("hed.configCompare" "minTraversal" 'boolean t)
```

Related Topics

[Compare Configuration Variables](#)

subConfigs

```
hed.configCompare subConfigs boolean { t | nil }
```

Description

Specifies that the traversal results of subconfigurations should also be included in the output file and ignores the subconfiguration traversal results. The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("hed.configCompare" "subConfigs")  
envSetVal("hed.configCompare" "subConfigs" 'boolean t)
```

Related Topics

[Compare Configuration Variables](#)

diffTool

```
hed.configCompare diffTool string "toolname"
```

Description

Specifies the tool to be used to compare the traversal results. The default comparison tool is `tkdiff`.

GUI Equivalent

None

Examples

```
envGetVal("hed.configCompare" "diffTool")  
envSetVal("hed.configCompare" "diffTool" 'string "tkdiff -w")
```

Related Topics

[Compare Configuration Variables](#)

Pin Check Variables

This section provides information on the names, descriptions, and graphical user interface equivalents for the Compare Configuration environment variables.

You can use these environment variables to control the traversal output displayed while performing the configuration traversal and comparison:

- pcLogFileOpenMode
- pcLogFilePath

pcLogFileOpenMode

```
pinCheck pcLogFileOpenMode cyclic { "append" | "overwrite" }
```

Description

Stores the pin checker logs in the log file.

- **append:** Appends the new pin checking results in the log file. This is the default.
- **overwrite:** Adds the latest pin checking result in the log file and removes the previous pin checking results from the log file.

GUI Equivalent

None

Examples

```
envGetVal("pinCheck" "pcLogFilePath")  
envSetVal("pinCheck" "pcLogFilePath" 'cyclic "overwrite")
```

Related Topics

[Log Files Creation](#)

pcLogFilePath

```
pinCheck pcLogFilePath string "file path"
```

Description

Changes the log file creation path.

The default value is the current working directory.

GUI Equivalent

None

Examples

```
envGetVal("pinCheck" "pcLogFilePath")  
envSetVal("pinCheck" "pcLogFilePath" 'string  
"pincheck_amsPLL_PLL_160MHZ_sim_config.log")
```

Related Topics

[Log Files Creation](#)

Hierarchy Editor Forms

This section lists the forms that can either be invoked only through the Hierarchy Editor or forms that can also be invoked outside but have options that are related to the Hierarchy Editor tool.

[Compare Config Form](#)

[Edit Constants Form](#)

[Explain Form \(Selected Cell\)](#)

[Explain Form \(Selected Instance\)](#)

[Explain Form \(Selected Occurrence\)](#)

[Filters Form](#)

[New Configuration Form](#)

[Save As Verilog Form](#)

[Save As VHDL Form](#)

[Open Form](#)

[Options Form \(Cell Table\)](#)

[Options Form \(Instance Table\)](#)

[Options Form \(Tree\)](#)

[Options Form \(General\)](#)

[Options Form \(Checks\)](#)

[Pin Checker Options Form](#)

Virtuoso Hierarchy Editor User Guide

Hierarchy Editor Forms

[Traverse config Form](#)

[Virtuoso Hierarchy Editor Editing Form](#)

[Virtuoso Hierarchy Editor Editing Form \(Table View\)](#)

[Virtuoso Hierarchy Editor Editing Form \(Tree View\)](#)

Add Occurrence Binding Form

Use this form to change or add a library or cell to the occurrence binding.

Field	Description
<i>Occurrence Binding</i>	Displays the details to update the occurrence binding.
<i>Library</i>	Specifies the library to which you want to bind the occurrence.
<i>Cell</i>	Specifies the cell to which you want to bind the occurrence.
<i>View</i>	Specifies the view o which you want to bind the occurrence.

Related Topics

[Defining Occurrence Bindings](#)

Compare Config Form

Use this form to compare two configurations and highlight their differences by opening the resulting design hierarchy side-by-side in a tree view.

Field	Description
First Config	Specifies the details of first configuration.
<i>Library</i>	Provides a longer description for your bookmark. This text appears in the <i>Description</i> column of the Bookmarks Manager window.
<i>Cell</i>	Adds the bookmark to the <i>Bookmarks</i> toolbar.
<i>View</i>	Bookmarks all the open tabs in your session window as a composite bookmark.
<i>Save traversal results to the specified file</i>	<p>Specifies an output file in which traversal results are saved. If you do not select this option, the traversal files are saved in the default traversal output directory named <code>configTraversals</code>. The traversal files are saved with the following naming convention:</p> <p><code><lib>.<cell>.<view></code></p> <p>For example, <code>tdmsDemoLib.ccadc.config</code></p> <p>where, <code>tdmsDemoLib</code> is a library, <code>ccadc</code> is a cell, and <code>config</code> is a view.</p>
<i>Output file</i>	Browses and selects the output file in which you want to save the results. If the output file already exists, a warning message appears to confirm overwriting the file.
<i>Use existing traversal file as an input</i>	Performs the traversal for the configuration and want to provide an existing traversal file as an input in the current comparison. When you select this option, all the above described options are disabled.
<i>Input file</i>	<p>Browses and selects the existing traversal file to be used for the comparison. You can directly provide the path to the output file in this field that is to be used as an input file in the comparison.</p> <p>If you have already opened a configuration, the configuration information is populated automatically in the <i>First Config</i> section.</p>
Second Config	Specifies the details of second configuration.

Virtuoso Hierarchy Editor User Guide

Hierarchy Editor Forms

Field	Description
<i>Library</i>	Provides a longer description for your bookmark. This text appears in the <i>Description</i> column of the Bookmarks Manager window.
<i>Cell</i>	Adds the bookmark to the <i>Bookmarks</i> toolbar.
<i>View</i>	Bookmarks all the open tabs in your session window as a composite bookmark.
<i>Save traversal results to the specified file</i>	<p>Specifies an output file in which traversal results are saved. If you do not select this option, the traversal files are saved in the default traversal output directory named <code>configTraversals</code>. The traversal files are saved with the following naming convention:</p> <pre><lib>.<cell>.<view></pre> <p>For example, <code>tdmsDemoLib.ccadc.config</code></p> <p>where, <code>tdmsDemoLib</code> is a library, <code>ccadc</code> is a cell, and <code>config</code> is a view.</p>
Common Options	Specifies the common options for both configurations.
<i>Space Indentation</i>	<p>Specifies the number of spaces to be used for indentation. This help improve the readability of the output files.</p> <p>Default value: 2</p> <p>Value range: 1 to 8</p>
<i>Expand Subconfigs</i>	Traverses HED through the sub-configurations found in the configuration. This field is not selected by default.
<i>Full traversal</i>	Performs a full traversal, which means when a cell is instantiated several times, each instance of this cell is fully traversed and a tree view for each instance of this cell is written out to an output file. If you do not select this option, only the first visited instance of this cell is fully traversed. This field is selected by default.
<i>Start diff tool</i>	Specifies a tool that you want to use to compare the traversal results. By default, the <code>tkdiff</code> tool is used for comparison.
<i>Output directory</i>	Browses and selects an output directory in which the traversal results are saved. If you do not specify anything, the <code>configTraversal</code> directory, which is present in the directory from where you launch HED, is used to save the traversal files.

Virtuoso Hierarchy Editor User Guide

Hierarchy Editor Forms

Field	Description
<i>MaskLayout stop limit (MB)</i>	Specifies the limit value in MB for the <code>maskLayout</code> views.

Related Topics

[Comparing Configurations](#)

Edit Constants Form

Use this form to create constants and edit the views included in an existing constants.

Field	Description
<i>Constant</i>	Represents a view list.
<i>Value</i>	Contains the list of views that the constant represents. Either type the view names or select them from the <i>Views</i> list box. Views must be listed in order of preference—the Hierarchy Editor uses the first view that is found.
<i>Views</i>	Contains the list of views from which you select views for the constant. You can customize the <i>View Choices</i> list.
<i>Insert</i>	Adds the selected view to the constant.
<i>Remove</i>	Removes views from the constant.
<i>Set</i>	Assigns the views to the constant and leaves the form open.
<i>Delete</i>	Removes the selected constant name and its values.
<i>Close</i>	Closes the form and saves your new or edited constants for the current session only.

Related Topics

[Changing the Views in the View Choices List Box](#)

Edit Description Form

Use this form to edit the description of your configuration..

Field	Description
<i>Description</i>	Displays the current description of the configuration, which you can edit.

Related Topics

[Viewing the Description File](#)

Explain Form (Selected Cell)

Use this form to view which binding rules were used for a cell listed in the *Cell Bindings* table.

Field	Description
<i>Selection</i>	Displays the master cellview for the selected cell. If the <i>Library</i> , <i>Cell</i> , or <i>View</i> fields display **UNBOUND** , the cell has a bind-to-open attribute.
<i>Library</i>	Specifies the master library of the selected cell.
<i>Cell</i>	Specifies the master cell of the selected cell.
<i>View</i>	Specifies the master view of the selected cell.
<i>Instantiations</i>	Lists all the instantiations of the cell. You can also open the parent cellview of the selected instance by clicking the Right Mouse Button and selecting <i>Open Parent</i> from the context-sensitive menu. This option allows opening the parent cellview of the selected instance (out of context) or the selected occurrence in the context of the current configuration.
<i>Explanation</i>	Describes the bindings for the selected cell.

Related topics

[Explain Form \(Selected Occurrence\)](#)

[Explain Form \(Selected Instance\)](#)

[Verifying Binding Rules](#)

Explain Form (Selected Instance)

Use this form to view which binding rules were used for an instance listed in the *Instance Bindings* table.

Field	Description
Selection	Displays the cellview that contains the selected instance
<i>Library</i>	Specifies the library of the cellview that contains the selected instance.
<i>Cell</i>	Specifies the cell of the cellview that contains the selected instance.
<i>View</i>	Specifies the view name of the cellview that contains the selected instance.
<i>Instance</i>	Specifies the name of the instance in the cellview that contains it.
Bound To	Displays the results of the rules that are used by the selected instance.
<i>Library</i>	Specifies the master library of the selected instance. If the <i>Library</i> , <i>Cell</i> , or <i>View</i> fields display **UNBOUND** , the instance has a bind-to-open attribute.
<i>Cell</i>	Specifies the master cell of the selected instance.
<i>View</i>	specifies the master view of the selected instance.
<i>Explanation</i>	Describes the bindings for the selected instance.

Related topics

[Explain Form \(Selected Cell\)](#)

[Explain Form \(Selected Occurrence\)](#)

[Verifying Binding Rules](#)

Explain Form (Selected Occurrence)

Use this form to view which binding rules were used for an occurrence.

Field	Description
Selection	Displays the cellview that contains the selected occurrence.
<i>Library</i>	Specifies the library of the cellview that contains the selected occurrence.
<i>Cell</i>	Specifies the cell of the cellview that contains the selected occurrence.
<i>View</i>	Specifies the view name of the cellview that contains the selected occurrence.
<i>Instance</i>	Specifies the name of the instance in the cellview that contains the occurrence.
Bound To	Displays the library, cell, and view to which the occurrence is bound. If the <i>Library</i> , <i>Cell</i> , or <i>View</i> fields display **UNBOUND** , the instance has a bind-to-open attribute.
<i>Library</i>	Specifies the library of the selected occurrence.
<i>Cell</i>	Specifies the master cell of the selected occurrence.
<i>View</i>	specifies the master view of the selected occurrence.
<i>Explanation</i>	Describes the bindings for the selected occurrence.

Related topics

[Explain Form \(Selected Cell\)](#)

[Explain Form \(Selected Instance\)](#)

[Verifying Binding Rules](#)

Filters Form

Use this form to specify filters which can help you to find cellviews quickly.

Field	Description
<i>Library</i>	<p>Specifies the search criteria for library names.</p> <p>For the <i>Library</i>, <i>Cell</i>, and <i>View</i> fields, you can:</p> <ul style="list-style-type: none">■ Specify whole words or character strings■ Specify multiple patterns; separate the patterns with spaces■ Use the following wildcards:<ul style="list-style-type: none"><input type="checkbox"/> <code>?</code>: Matches any single character<input type="checkbox"/> <code>[list]</code>: Matches any single character in list<input type="checkbox"/> <code>[lower-upper]</code>: Matches any character in the range between lower and upper<input type="checkbox"/> <code>*</code>: Matches any pattern <p>Pattern matching is case sensitive.</p>
<i>Cell</i>	Specifies the search criteria for cell names.
<i>View</i>	Specifies the search criteria for view names.
<i>Show leaf instances and cellviews</i>	Specifies whether you want to display instances and cellviews that are at the leaf level of the tree, that is, they do not have any items under them.
<i>Show instances and cellviews that have invalid bindings</i>	Specifies whether you want to display instances and cellviews that are unbound or whose binding is invalid.
<i>Only show entries that have an explicit rule</i>	Specifies whether you want to display all entries or only those entries for which a view or view list has been set explicitly in the <i>View to Use</i> or <i>Inherited View List</i> column respectively (that is, the inherited values have been overridden). Entries with an explicit Bind-to-Open, Source File, or Reference Verilog setting are also displayed when this option is selected.
<i>Defaults</i>	Restores the default match criteria and turns the filter off. The Hierarchy Editor displays all cellviews and instances. The status bar displays <i>Filters OFF</i> .

Virtuoso Hierarchy Editor User Guide

Hierarchy Editor Forms

Related topics

[Filtering Cellviews](#)

New Configuration Form

Use this form to create and use a configuration with the Hierarchy Editor.

Field	Description
Top Cell	Specifies the top cellview of the design.
<i>Library</i>	Specifies the name of the library that contains the top cellview.
<i>Cell</i>	Specifies the cell name of the top cellview.
<i>View</i>	Specifies the view name of the top cellview.
Global Bindings	Specifies default bindings for the entire design.
<i>Library List</i>	Specifies libraries for cells that do not have fixed library bindings. List the libraries in the order you want them searched. Separate entries with spaces.
<i>View List</i>	Specifies the views you want in your configuration in order of preference. The view list applies to every level of the configuration and determines which view is selected for every object in the design, unless overridden by a cell or instance binding. Separate entries with spaces.
<i>Stop List</i>	Specifies the views that are to be treated as leaf nodes, that is, they are not to be expanded. Separate entries with spaces. This field can be empty.
<i>Constraints List</i>	Specifies a list of constraint views. The first view that is found is used. If none of the views in the list are found, the schematic editor creates a new view, of the same name as the first view in the list.
<i>Description</i>	Lets you enter a brief statement describing the configuration.
<i>Use Template</i>	Lets you select a template from a list of predefined templates.

Related topics

[Creating a New Configuration](#)

[Wildcards in a View List](#)

[Changing the Views in the View Choices List Box](#)

Virtuoso Hierarchy Editor User Guide

Hierarchy Editor Forms

Working with Templates in Hierarchy Editor

Save As Form

Use this form to save a configuration for VHDL applications.

Field	Description
<i>Configuration</i>	Selects the library in which you want to save the VHDL configuration.
<i>Library</i>	Specifies the name of the library in which you want to save a copy of the configuration.
<i>Cell</i>	Specifies the name of the cell in which you want to save the configuration.
<i>View</i>	Specifies the name of the new configuration view.
<i>Check for Verilog</i>	If the <i>Check for Verilog</i> option is deselected, the Hierarchy Editor adds all cellviews that do not have VHDL source files to the configuration but comments them out and generates a warning for them in the Save as VHDL Warnings dialog box.

Related topics

[Saving a Configuration](#)

Save As Verilog Form

Use this form to save a configuration for Verilog applications.

Field	Description
<i>Look in</i>	Selects the directory in which you want to save the configuration.
<i>File</i>	Specifies a name for the Verilog file. The default name for the Verilog file is <code>verilog.f</code> .
<i>Files of type</i>	Narrows the list of files that are displayed by choosing to display only certain types of files.
<i>Save</i>	Saves the Verilog file and closes the form.

Related topics

[Saving a Configuration](#)

Save As VHDL Form

Use this form to save a configuration for VHDL applications.

Field	Description
<i>Configuration</i>	Selects the library in which you want to save the VHDL configuration.
<i>Library</i>	Selects the cell in which you want to save the VHDL configuration.
<i>Cell</i>	Represents the configuration view. For VHDL configurations, the view name is always <code>configuration</code> .
<i>View</i>	Adds cellview that do not have VHDL source files but do have Verilog source files to the VHDL configuration. Select this option if you want to add these cellviews to the VHDL configuration; deselect it if you do not want to add these cellviews to the VHDL configuration.
<i>Check for Verilog</i>	If the <i>Check for Verilog</i> option is deselected, the Hierarchy Editor adds all cellviews that do not have VHDL source files to the configuration but comments them out and generates a warning for them in the Save as VHDL Warnings dialog box.

Related topics

[Saving a Configuration](#)

Open Form

Use this form to open a configuration from the Hierarchy Editor.

Field	Description
<i>Configuration</i>	Specifies the library, cell, and view details for which you want to open a configuration.
<i>Library</i>	Selects the name of the library that contains the design. The drop-down list displays all the libraries that are specified in your <code>cds.lib</code> file.
<i>Cell</i>	Selects the name of the cell that contains the configuration view. The drop-down list displays all the cells in the library you selected.
<i>View</i>	Selects the name of the configuration view you want to open. The drop-down list contains all the configuration views that are in the cell you selected.

Related topics

[Opening Configurations in Hierarchy Editor](#)

Options Form (Cell Table)

Use this form to specify the attributes you want to display in the cell table, instance table, and tree view of the Hierarchy Editor. It also lets you select the automatic update option.

Use this tab of the Options Form to select the columns that appear in the cell table.

Field	Description
Show Cell Columns	Specifies the cell column details.
<i>Info</i>	Specifies the library, cell, and view details for which you want to open a configuration. For example, a green pyramid in this column indicates the top-level cell view. For a complete list of icons that can be displayed in this column and their meanings, choose <i>Help – Legend</i> to display the <u>Legend dialog box</u> .
<i>View to Use</i>	Displays a field in which you can type a specific view to use for the cell.
<i>Inherited View List</i>	Displays a field in which you can modify the inherited view list for the cell. The default inherited view list is the global view list.
<i>Inherited Lib List</i>	Displays a field in which you can modify the inherited library list for the cell. The default inherited library list is the global library list.

Related topics

[Options Form \(Instance Table\)](#)

[Options Form \(Tree\)](#)

[Options Form \(General\)](#)

[Options Form \(Checks\)](#)

Options Form (Instance Table)

Use this tab of the Options Form to select the columns that appear in the instance table..

Field	Description
Show Instance Columns	Specifies the instance column details.
<i>Info</i>	Displays information about the cellview. For example, a green pyramid in this column indicates the top-level cell view. For a complete list of icons that can be displayed in this column and their meanings, choose <i>Help – Legend</i> to display the <u>Legend dialog box</u> .
<i>View to Use</i>	Displays a field in which you can type a specific view to use for the instance.
<i>Inherited View List</i>	Displays a field in which you can modify the inherited view list for the instance.
<i>Inherited Lib List</i>	Displays a field in which you can modify the inherited library list for the instance.

Related topics

[Options Form \(Cell Table\)](#)

[Options Form \(Tree\)](#)

[Options Form \(General\)](#)

[Options Form \(Checks\)](#)

Options Form (Tree)

Use this tab of the Options Form to select the columns that appear in the tree structure..

Field	Description
Show Tree Columns	Specifies the library, cell, and view details for which you want to open a configuration.
<i>View to Use</i>	Displays displays a field in which you can type a specific view to use for the instance or occurrence.
<i>Inherited View List</i>	Displays a field in which you can modify the inherited view list for the instance or occurrence.
<i>Inherited Lib List</i>	Displays a field in which you can modify the inherited library list for the instance or occurrence.
Expand Mode	Specifies the details to display while expanding the tree.
<i>By Instance</i>	Displays all the instances for the selected cell when you expand a tree node.
<i>By Instance Grouping</i>	Displays similar instances grouped together when you expand a tree node. The number of instances is also displayed if the <i>Show instance details</i> option has been selected. The default expand mode is <i>By instance</i> .
Display Details	Specifies the highlight and display details.
<i>Highlight expanded rows</i>	Highlights instances in expanded tree nodes.
<i>Show instance details</i>	Displays the library-cell-view to which the instance is bound, in addition to the instance name. For example: <i>Q1(tdmsDemoLib bmen spectreS)</i> . It also displays the number of instances for the cell if the <i>Expand Mode</i> is set to <i>By Instance Grouping</i> .

Related topics

[Options Form \(Cell Table\)](#)

[Options Form \(Instance Table\)](#)

[Options Form \(General\)](#)

[Options Form \(Checks\)](#)

Options Form (General)

Use this tab of the Options Form to specify the general settings.

Field	Description
Update	Specifies the update details.
<i>Auto Update</i>	Updates the configuration automatically every time you make a change. If this option is selected, you do not have to click the <i>Update</i> icon in the toolbar or choose <i>View – Update</i> to recompute the hierarchy or view your changes.
<i>On update query and / or save modified cellviews</i>	Check this option to display the updateSync form. Unchecking this option means that this form is not displayed when an update is performed. updateSync is only required when you want to synchronize changes to the configuration with an external process.
Properties	Specifies the property details.
<i>Properties are inheritable by default</i>	Sets the properties that are not inherited down the hierarchy.
Display	Specifies the display details.
<i>Auto resize columns to fit</i>	Resizes the columns. If a window is reduced in size, a scrollbar is added. By default, columns are auto-resized to fit the window. Highlights instances in expanded tree nodes.
<i>Broadcast multiple selections</i>	Broadcasts all the selected items to the Virtuoso Schematic Editor. By default, only single selections are broadcast. For example, if the top cell schematic is opened “in-context”, and multiple instances are selected in the Hierarchy Editor, then only the last selection is cross-selected in the schematic.
<i>Evaluate Pcells</i>	Performs the Pcell evaluation. Choosing to enable Pcell evaluation instructs the Hierarchy Editor to perform a more complete elaboration of the design hierarchy.
ADE	Specifies the ADE details.

Virtuoso Hierarchy Editor User Guide

Hierarchy Editor Forms

Field	Description
<i>Set simulator to AMS when launching ADE</i>	<p>Sets the simulator to AMS. If not selected, HED does not change the simulator and the current ADE startup simulator <code>.cdsenv</code> value gets considered.</p> <pre>hed.ade setAmsSimulator boolean t/nil</pre> <p>If <code>t</code> (default) is specified, the simulator value is set to "ams".</p> <p>If <code>nil</code> is specified, the simulator value does not get modified by HED and selects the value based on the ADE startup simulator <code>.cdsenv</code> value.</p>
<i>Defaults</i>	Resets the values to their defaults.

Related topics

[Options Form \(Cell Table\)](#)

[Options Form \(Instance Table\)](#)

[Options Form \(Tree\)](#)

[Options Form \(Checks\)](#)

Options Form (Checks)

Use this tab of the Options Form to specify the checks to run and when to run them. The checks can be run when opening a configuration, updating the configuration, or saving the configuration.

Field	Description
<i>Snapshot config objects</i>	Checks if the snapshot config objects have an appropriate binding to prevent the library or view list from being used while opening, updating, or saving the config.
<i>Libraries</i>	Check if libraries are valid in the global and inherited library list while opening, updating, or saving the config.
<i>Editing a config</i>	Checks for unused config rules that need to be removed while opening, updating, or saving the config.
<i>Defaults</i>	Resets the values to their defaults.

Related topics

[Options Form \(Cell Table\)](#)

[Options Form \(Instance Table\)](#)

[Options Form \(Tree\)](#)

[Options Form \(General\)](#)

Package Checking Viewer Form

Use this form to view a list of imported packages, comprising VHDL and SystemVerilog (SV) modules and their dependency relationships.

Tab	Description
<u>Table View</u>	Displays the packaging details in table form.
<u>Tree View</u>	Displays the packaging details in tree form.

Table View

The following table describes the fields available on the Table view tab of the Package Checking Viewer form.

Field	Description
<i>Refresh</i>	Refreshes the list of packages in the <i>Package Summary</i> area.
<i>Full List</i>	Displays the built-in packages in the <i>Package Summary</i> area. The source file and package dependents are not displayed in the <i>Package Summary</i> area.
<i>List Details</i>	Displays the associated source files log and dependents' list of the selected package.
<i>Package Summary</i>	Displays the list of packages.
<i>Package Details</i>	Displays the source file and dependents' list of the selected package.

Tree View

The following table describes the fields available on the Tree View tab of the Package Checking Viewer form.

Field	Description
<i>Package Dependent</i>	Displays a list of packages and their dependency relationships.

Virtuoso Hierarchy Editor User Guide

Hierarchy Editor Forms

Related topics

[Checking Imported Packages](#)

Pin Checker Options Form

Use this form to detect mismatch in the connectivity and to fix the issues before netlisting..

Field	Description
<i>Dynamic pin check</i>	<p>Performs pin checking automatically if the view of a cell or an instance gets changed using the RMB options. The updated value is applicable to all the open HED windows. By default, <i>Dynamic pin check</i> is selected in the Pin Checker Options form.</p> <p>You can change the view of a cell, right-click the cell in the <i>Table View</i> and select the <i>Set Cell View</i> option from the context-sensitive menu. Similarly, to change the view of an instance, right-click the instance either in the <i>Table View</i> or <i>Tree View</i> and select the <i>Set Instance View</i> option from the context-sensitive menu.</p> <p>If <i>Dynamic pin check</i> is not selected, you need to explicitly select the <i>Pin Check Cell Instantiations</i> or <i>Pin Check Instance</i> option from the context-sensitive menu, if the view of that cell or instance is changed.</p>
<i>AMS-UNL checks</i>	<p>Resolves the issues related to the mismatch of vector and split ports on the place master and switch master in Pin Checker. These checks are already implemented in AMS-UNL netlister.</p> <p>This command works only if the instance parent cellview is non-text and the switch master cellview is text.</p> <p>When this option is selected, the Pin Checker does not reports an error when there is a mix of vector and split ports in the place master and switch master. This check is a window specific check. Selecting this field only effects the HED window.</p>

Virtuoso Hierarchy Editor User Guide

Hierarchy Editor Forms

Field	Description
<i>HDL package setup</i>	<p>Imports all of the settings automatically from the HDL Package Setup form. This functionality is available only if the path to the file containing the settings has been specified in the HDL Package Setup.</p> <p>The settings in the HDL Package Setup form must be saved after ensuring that the packages have been compiled using the specified options. If not then the compiler options from the HDL Package Setup form are found in the HDL Package Setup form or if the HDL package setup check box is not selected, the Pin Checker continues with the previous setup.</p>

Related topics

[Pin Checker](#)

Populate Library with Verilog Modules Form

Use this form to populate a library with Verilog modules.

Field	Description
From	Specifies the source files of the Verilog modules
<i>Files</i>	<p>Specifies the paths to the files.</p> <p>You can use an asterisk (*) as a wildcard in the paths you specify. For example: <code>/myProject/verilog/*.vams</code> gets modules from all files in the <code>verilog</code> directory that have a <code>.vams</code> extension and <code>/myproject/verilog/*</code> gets modules from all the files in the directory.</p>
<i>Browse</i>	Selects the files one at a time.
To	Specifies the destination files of the Verilog modules
<i>Library</i>	<p>Selects the library to which you want to add the Verilog modules. The drop-down list displays all the libraries that are in your library definition file (<code>cds.lib</code>).</p> <p>You must have write access to the master library or its temporary directory.</p>
<i>View</i>	Specifies a name for the Verilog views. You can choose any name; the default name is <code>module</code> .
<i>Update library list</i>	<p>Adds the selected library to the global library list of the configuration. If you select this option, the library is added as the first library in the library list.</p> <p>If a configuration is not open, or if the library is already in the global library list of the configuration, this field is grayed-out.</p>
<i>Update view list</i>	<p>Adds the specified Verilog views to the global view list of the configuration. If you select this option, the view is added as the first view in the global view list.</p> <p>If a configuration is not open, or if the view is already in the global view list of the configuration, this field is grayed-out.</p>

Related topics

[Using Verilog Files in Configuration](#)

Reference Verilog Modules Form

Use this form to reference a Verilog file for a cell, instance, or occurrence.

Field	Description
From	Specifies the source file of the Verilog modules.
<i>File</i>	Specifies the path to the file.
<i>Browse</i>	Selects the files one at a time. The Verilog file you select must have one of the following extensions: <code>.vams</code> or <code>.v</code> because most Cadence applications recognize only files with these extensions as Verilog files.
To	Specifies the destination files of the Verilog modules.
<i>Library</i>	Selects the library in which you want to place the cellviews when the Verilog file is compiled. The cellviews are present in the implicit temporary directory, not in the master library. If the library is fixed by some other rules in the design data, you won't be able to edit the <i>Library</i> field. For example, design data created by the Virtuoso schematic editor typically does not allow you to change the library.
<i>View</i>	Specifies a name for the Verilog view. You can choose any name; the default name is <code>module</code> . The view gets implemented later with the modules in the Verilog file by other applications in the flow. For example, in the design preparation step, AMS creates cellviews for the modules in the implicit temporary directory.
<i>Update library list</i>	Adds the specified library to the inherited library list of the cell, instance, or occurrence. If you select this option, the library is added as the first library in the inherited library list. If the library is already in the inherited library list, this field is grayed-out.
<i>Update view list</i>	Adds the specified views to the inherited view list for the cell, instance, or occurrence. If you select this option, the view is added as the first view in the inherited view list. If the view is already in the inherited view list, this field is grayed-out.

Related topics

[Using Text Files in Configuration](#)

[Using Verilog Files in Configuration](#)

[Populate Library with Verilog Modules Form](#)

Traverse config Form

Use this form to traverse through a configuration and save the traversal results in an output file.

Field	Description
Config	Specifies the details of first configuration.
<i>Library</i>	Specifies the library for the first configuration.
<i>Cell</i>	Specifies the cell.
<i>View</i>	Specifies the view.
<i>Save traversal results to the specified file</i>	<p>Specifies an output file in which traversal results are saved. If you do not select this option, the traversal files are saved in the default traversal output directory named <code>configTraversals</code>. The traversal files are saved with the following naming convention:</p> <p><code><lib>.<cell>.<view></code></p> <p>For example, <code>tdmsDemoLib.ccadc.config</code></p> <p>where, <code>tdmsDemoLib</code> is a library, <code>ccadc</code> is a cell, and <code>config</code> is a view.</p>
<i>Output file</i>	<p>Browses and selects the output file in which you want to save the results. If the output file already exists, a warning message appears to confirm overwriting to this file. This option is available only if you select the <i>Save traversal results to the specified file</i> checkbox.</p>
<i>Save traversal results to the specified file</i>	<p>Selects this checkbox if you want to specify an output file in which traversal results are saved. If you do not select this option, the traversal files are saved in the default traversal output directory named <code>configTraversals</code>. The traversal files are saved with the following naming convention:</p> <p><code><lib>.<cell>.<view></code></p> <p>For example, <code>tdmsDemoLib.ccadc.config</code></p> <p>where, <code>tdmsDemoLib</code> is a library, <code>ccadc</code> is a cell, and <code>config</code> is a view.</p>
Common Options	Specifies the common options for both configurations.

Virtuoso Hierarchy Editor User Guide

Hierarchy Editor Forms

Field	Description
<i>Space Indentation</i>	Specifies the number of spaces to be used for indentation. This help improve the readability of the output files. Default value: 2 Value range: 1 to 8
<i>Expand Subconfigs</i>	Traverses HED through the sub-configurations found in the configuration. This field is not selected by default.
<i>Full traversal</i>	Performs a full traversal, which means when a cell is instantiated several times, each instance of this cell is fully traversed and a tree view for each instance of this cell is written out to an output file. If you do not select this option, only the first visited instance of this cell is fully traversed. This field is selected by default.
<i>Output directory</i>	Browse and select an output directory in which the traversal results are saved. If you do not specify anything, the <code>configTraversal</code> directory, which is present in the directory from where you launch HED, is used to save the traversal files.
<i>MaskLayout stop limit (MB)</i>	Specifies the limit value in MB for the <code>maskLayout</code> views.

Related Topics

[Traversing Configurations](#)

Use Template Form

Use this form to specify the template to use to create a new configuration.

Field	Description
Template	Specifies the template details you want to use.
<i>Name</i>	Lets you select a template from a list of pre-defined templates for simulators that are compatible with Cadence software. The listbox also lists any other templates that you placed in the <code>your_install_dir/share/cdssetup/hierEditor/templates/</code> directory. If you want to use a template that is not in this directory, select <i><Other></i> and specify the path in the <i>From File</i> field.
<i>From File</i>	Lets you type the path to the template you want to use.

Related Topics

[Working with Templates in Hierarchy Editor](#)

View List Building Forms

Use this tab to modify global bindings in HED, saving on the need to manually create global binding lists.

Field	Description
<i>Library List</i>	Displays the Edit Library List form.
<i>View List</i>	Displays the Edit View List form.
<i>Stop List</i>	Displays the Edit Stop List form.
<i>Constraint List</i>	Displays the Edit Constraint List form.
<i>Add</i>	Adds items to the selected list.

Related Topics

[Virtuoso Hierarchy Editor Editing Form](#)

[Virtuoso Hierarchy Editor Editing Form \(Table View\)](#)

[Virtuoso Hierarchy Editor Editing Form \(Tree View\)](#)

[Displaying the Table View in Hierarchy Editor](#)

Virtuoso Hierarchy Editor Editing Form

Use this form to create a configuration that provides expansion information for mixed-signal partitioning. This form lets you view many levels of a single design using either a table view or a tree view..

Field	Description
<i>Top Cell</i>	Contains the library, cell, and view names of the top cellview of the configuration you opened. This is the root cellview of your design.
<i>Library</i>	Displays library name of the top cellview of the configuration
<i>Cell</i>	Displays cell name of the top cellview of the configuration.
<i>View</i>	Displays view name of the top cellview of the configuration.
<i>Opens</i>	Opens the top cellview in an editor. If the top cellview is a schematic, it is opened in the Virtuoso® schematic editor; if it is a Verilog or VHDL view, it is opened in a text editor.
<i>Edit</i>	Opens the Edit Top Cell form where you can select alternative <i>Library</i> , <i>Cell</i> , and <i>View</i> selections for the Top Cell from drop-down menus.
<i>ADE L</i>	Opens the ADE L window.
<i>ADE Explorer</i>	<p>Opens an existing ADE Explorer view or creates a new ADE Explorer view.</p> <ul style="list-style-type: none"> ■ When opening an existing ADE Explorer view, the simulator is the same as previously set in the cellview. ■ When creating a new ADE Explorer view, the simulator is set to <code>ams</code> by default.
<i>Global Bindings</i>	<p>Contains the global library, view, and stop lists. These are the default values for the entire configuration.</p> <p>Selecting the browse (...) button for a specific <i>Global Bindings</i> list option displays a view list building form that allows you to build up a view list without the need to manually type in the libraries, views, and so on.</p>

Virtuoso Hierarchy Editor User Guide

Hierarchy Editor Forms

Field	Description
<i>Library List</i>	Displays the list of libraries, specified in order of preference, that are searched for cells. Each cell is obtained from the first library in the list that contains a cell of that name.
<i>View List</i>	Displays the list of views, specified in order of preference, for cells. For each cell, the first view in the list that is found is used. For example, if the view list is <code>tbench_1 schematic dataflow</code> , the netlister first looks for a <code>tbench_1</code> view. If it is not found, it looks for a <code>schematic</code> view. If that is not found either, it looks for a <code>dataflow</code> view.
<i>Stop List</i>	Displays the list of views used to determine when the hierarchy expansion stops and when a cell should be considered a leaf node in the hierarchy.
<i>Constraint List</i>	Displays the list of constraint views, listed in order of preference. The first view that is found is used. If none of the constraint views in the list are found, the schematic editor creates an empty view named after the first view in the constraint list.
<i>Table View</i>	Displays all the cells of your design in a table format.
<i>Tree View</i>	Displays the hierarchy of your design. It displays the top cellview and all the instances contained in it.

Related Topics

[Displaying the Table View in Hierarchy Editor](#)

[Virtuoso Hierarchy Editor Editing Form \(Table View\)](#)

[Virtuoso Hierarchy Editor Editing Form \(Tree View\)](#)

Virtuoso Hierarchy Editor Editing Form (Table View)

Use this tab to view various levels of a single design using the table view..

Field	Description
Cell Bindings	Displays the library and view bindings of cells.
<i>Library</i>	Specifies the library name.
<i>Cell</i>	Specifies the cell name of the given library.
<i>View Found</i>	Displays the view that is found for the cell based on the binding rules.
<i>View to Use</i>	Displays a field in which you can type a specific view to use for the cell.
<i>Inherited View List</i>	Displays a field in which you can modify the inherited view list for the cell. The default inherited view list is the global view list.
Instance Bindings	Displays all the instances in the block you selected and their bindings
<i>Library</i>	Specifies the library name.
<i>Cell</i>	Specifies the cell name of the given library.
<i>View Found</i>	Displays the view that is found for the instance based on the binding rules.
<i>View to Use</i>	Specifies a specific view for the instance.
<i>Inherited View List</i>	Displays a field in which you can modify the inherited view list for the cell. The default inherited view list is the global view list.

Related Topics

[Displaying the Tree View in Hierarchy Editor](#)

[Virtuoso Hierarchy Editor Editing Form](#)

[Virtuoso Hierarchy Editor Editing Form \(Tree View\)](#)

Virtuoso Hierarchy Editor Editing Form (Tree View)

Use this tab to view various levels of a single design using the tree view..

Field	Description
<i>Instance</i>	Displays the instance name. The library-cell-view binding of the instance is listed in parenthesis next to the name.
<i>View to Use</i>	Specifies a specific view for the instance.
<i>Inherited View List</i>	Displays the inherited view list that determines the view binding of the instance, unless it is overridden by a View to Use value. You can edit this column.

Related Topics

[Virtuoso Hierarchy Editor Editing Form](#)

[Virtuoso Hierarchy Editor Editing Form \(Table View\)](#)

Hierarchy Editor SKILL Functions

This topic provides a list of Cadence® SKILL functions associated with Hierarchy Editor.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

- hedRegUICustomFunc
- hnlRunPinChecker

hedRegUICustomFunc

```
hedRegUICustomFunc (  
    s_uifunc  
)  
=> t / nil
```

Description

Registers a UI customization function that gets called when a new Hierarchy Editor window is created.

Arguments

<i>s_uifunc</i>	A UI customization function that accepts the <code>hiWindowId</code> value of the window being customized.
-----------------	--

Value Returned

<code>t</code>	The customization function was registered.
<code>nil</code>	Registration failed.

Related Topics

[hnlRunPinChecker](#)

hnlRunPinChecker

```
hnlRunPinChecker(  
    t_lib  
    t_cell  
    t_view  
    [ ?checkAms { t | nil } ]  
    [ ?useHDLPackageSettings { t | nil } ]  
    [ ?compilerOptionFile t_compilerOptionFile ]  
)  
=> t / nil
```

Description

Runs the Pin Checker tool from the SKILL interface without launching Virtuoso or the Hierarchy Editor. This is especially helpful when you want to automate the launch of Pin Checker functionality before netlisting or running any simulations.

Arguments

<i>t_lib</i>	The name of the library.
<i>t_cell</i>	The name of the config cell.
<i>t_view</i>	The name of the config cellview.
<i>?checkAms { t nil }</i>	Specifies whether the issues related to the mismatch of the vector and split ports on the place master and switch master must be resolved in the Pin Checker. The default is <i>nil</i> .

Virtuoso Hierarchy Editor User Guide

Hierarchy Editor SKILL Functions

`?useHDLPackageSettings { t | nil }`

Specifies whether the Pin Checker uses import settings from the HDL Package Setup form. The default is `nil`.

You can import settings from the HDL Package Setup form in the following ways:

- By automatically importing all settings from the HDL Package Setup form.

To do this, ensure that the packages have been compiled using the specified options in the HDL Package Setup form before the form is saved. HDL Package Setup settings can be imported only when these settings are saved in the current session of Virtuoso in which the Pin Checker has been launched. This means that both Pin Checker and HDL Package Setup form must be open in the current session of Virtuoso.

- By importing package settings in the Pin Checker without launching Virtuoso and without saving the HDL package Setup in each Virtuoso session.

To do this, set the `useHDLPackageSettings` argument to `t` and specify the settings that you need to compile the package in the HDL Package Setup form. Then, select the *Export XrunArgs* option to export the settings into a file. The Pin Checker can use this file through the `?compilerOptionFile` argument to import package settings without launching Virtuoso or the HDL Package Setup form.

`?compilerOptionFile t_compilerOptionFile`

The name of file saved from the HDL Package Setup form using the *Export XrunArgs* option. The default is " ".

Value Returned

<code>t</code>	No mismatch was found during the pin check.
<code>nil</code>	An error was reported while running the function or a mismatch was found during pin check.

Examples

The following example performs a pin check on the `config` cellview from the cell `topcell` within the `mylib` library using the Pin Checker tool.

```
hnlRunPinChecker("myLib" "topcell" "config")  
=> t
```

Related Topics

[Virtuoso HDL Package Form](#)

[hedRegUICustomFunc](#)