

Virtuoso Text Editor User Guide

**Product Version IC23.1
August 2023**

© 2023 Cadence Design Systems, Inc. All rights reserved.
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.

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

1

Introduction	7
<u>Licensing Requirements</u>	8
<u>Overview</u>	9
<u>Launching Virtuoso Text Editor</u>	9
<u>Understanding the Graphical User Interface</u>	11

2

Working With Text Cellviews	19
<u>Creating Cellviews</u>	20
<u>Creating a Text Cellview</u>	21
<u>Creating a Cellview from an Existing Cellview</u>	23
<u>Editing Text Cellviews</u>	26
<u>Switching Between Edit Mode and Read-Only Mode</u>	26
<u>Checking Syntax</u>	27
<u>Showing Line Numbers and Going to a Line Number</u>	28
<u>Editing a Text Cellview File in an External Editor</u>	30
<u>Checking Pins</u>	32
<u>Using Other Editing Features</u>	33
<u>Generating the Databases of Text Cellviews</u>	34
<u>Viewing the File Parse Log</u>	36
<u>Navigating a Design Hierarchy Containing Text Cellviews</u>	38
<u>Ignoring Explicitly Inherited Terminals for Text Cellviews</u>	41

A

Virtuoso Text Editor and HDL Package Setup Functions	43
<u>hdlPkgCreateEnvSetupFile</u>	44
<u>hdlPkgExportXrunArgs</u>	47
<u>teDiscardEdits</u>	51
<u>teGetCursorPosition</u>	52
<u>telsModified</u>	54
<u>teSave</u>	55
<u>teSaveWithDerivedData</u>	56
<u>teSetCursor</u>	57

Virtuoso Text Editor User Guide

<u>teSetEditMode</u>	58
<u>teRegPostExtractTrigger</u>	59
<u>teRegPreExtractTrigger</u>	61
<u>teUnRegPostExtractTrigger</u>	63
<u>teUnRegPreExtractTrigger</u>	64
<u>hdlGenerateTextDatabase</u>	65

B

Environment Variables 67

<u>CDS5X_NOLINK</u>	68
<u>enableAhdllintStaticCheck</u>	68
<u>hdlPkgAdvancedMode</u>	69
<u>hdlPkgEnableHdlSetup</u>	69
<u>hdlPkgEnableUPF</u>	70
<u>hdlPkgEnableUVM</u>	71
<u>hdlPkgHdlvarFile</u>	71
<u>hdlPkgMakeLibTree</u>	72
<u>hdlPkgProjDir</u>	73
<u>hdlPkgUseHdlvar</u>	73
<u>hdlPkgUseXmllibdirname</u>	74
<u>hdlPkgUVMVersion</u>	75
<u>hdlPkgXmllibdirname</u>	75
<u>hdlPkgXrunArgsFileTableList</u>	76
<u>inScopeReadBGColor</u>	77
<u>inScopeWriteBGColor</u>	77
<u>showLineNumbersInSideBar</u>	77
<u>syntaxLineLength</u>	78
<u>useExternalEditor</u>	78
<u>ignoreDesignValCheckXcelium</u>	79
<u>messageSeverityXcelium</u>	80
<u>designCheckLogFileXcelium</u>	81
<u>messageSeverityExtractionFramework</u>	82

C

<u>Virtuoso Text Editor Forms</u>	83
<u>New File Form</u>	84
<u>Open File Form</u>	86
<u>Cellview From Cellview Form</u>	87
<u>Find and Replace Form</u>	89
<u>Go to Line Form</u>	90
<u>Text Editor Options Form</u>	90
<u>Add Bookmark and Bookmark Manager Forms</u>	91
<u>Workspaces Forms</u>	91

D

<u>HDL Package Setup</u>	93
<u>Accessing the Virtuoso HDL Package Setup Form</u>	94
<u>Use of HDL Package Settings in Various Tools</u>	96
<u>Virtuoso HDL Package Setup Form</u>	99

Virtuoso Text Editor User Guide

Introduction

This document describes Virtuoso® Text Editor and helps you in using it to work with digital and analog text cellviews. It is intended for circuit designers who want to use Virtuoso Text Editor for working with design blocks stored in HDL files. This document also describes the SKILL functions of Virtuoso® Text Editor.

Note: Only the functions and arguments described in this manual are supported for public use. Any undocumented functions or arguments are likely to be private and could be subject to change without notice. It is recommended that you check with your Cadence representative before using them.



Important

Cadence provides example files that you can use to explore and implement Verifier SKILL functions. The examples, such as `test.il` and `verifier_email_example.il`, are located in the location `$CDSHOME/tools/dfII/samples/Verifier`.

The SKILL API reference is meant for verification project managers and designers who want to use Verifier SKILL APIs for requirements-based verification of their analog designs.

This document assumes that users are familiar with the Cadence SKILL™ language and Virtuoso Schematic Editor.

This document includes the following topics:

- [Overview](#) on page 9
- [Launching Virtuoso Text Editor](#) on page 9
- [Understanding the Graphical User Interface](#) on page 11
- [Working With Text Cellviews](#) on page 19
- [Virtuoso Text Editor and HDL Package Setup Functions](#) on page 43
- [Environment Variables](#) on page 67

- [Virtuoso Text Editor Forms](#) on page 83
- [Virtuoso Text Editor and HDL Package Setup Functions](#) on page 43

Licensing Requirements

Virtuoso Text Editor is available with Virtuoso Framework License 111. Virtuoso Schematic Editor L license 95100 is required to edit HDL files using Virtuoso Text Editor.

For more information on licensing, see [*Virtuoso Software Licensing and Configuration Guide*](#).

Overview

Virtuoso® Text Editor lets you work with the text cellviews stored in HDL files, such as Verilog, SystemVerilog, VHDL, PSpice, HSPICE, Spectre, and SPICE text files, in Virtuoso libraries. You use this application to perform the following tasks:

- Create and edit digital and analog text cellviews.
- Check the syntax of text cellviews.
- Create text cellview databases.
- Create different types of views, such as symbol and schematic views, from a text cellview.
- Check the pin order in text cellviews.

Virtuoso provides various tools to create text cellviews. For example, Virtuoso Verilog In lets you import modules into an external Verilog file as text cellviews in a Virtuoso library. You can continue to use these tools to create text cellviews, while using Virtuoso Text Editor to edit the text cellviews.

Launching Virtuoso Text Editor

You can launch Virtuoso Text Editor from various Virtuoso applications to open or create a text cellview.

The following table describes some methods to open a text cellview in Virtuoso Text Editor.

Virtuoso Application	Process to Launch Virtuoso Text Editor
Virtuoso Library Manager	➔ Double-click the text cellview.
Virtuoso CIW	<ol style="list-style-type: none">1. Choose <i>File — Open</i>. The Open File form appears.2. Select the text cellview and click <i>OK</i>. <p>For details, see <i>Working with Cellviews</i> in <i>Virtuoso Studio Design Environment User Guide</i>.</p>

Virtuoso Text Editor User Guide

Introduction

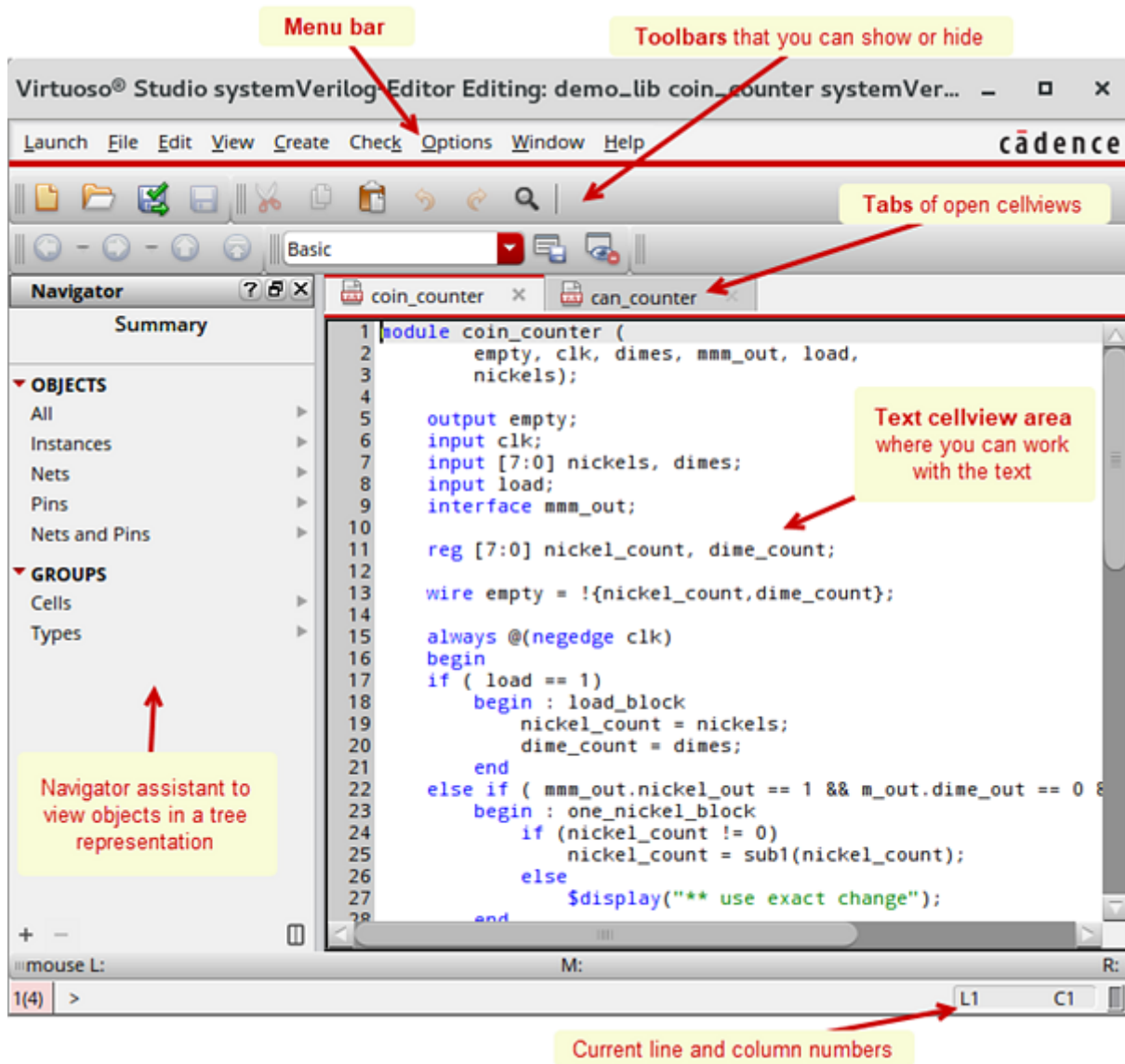
Virtuoso Application	Process to Launch Virtuoso Text Editor
Virtuoso Schematic Editor	<ol style="list-style-type: none">1. Open a design and double-click the symbol associated with a text cellview. The Descend form appears.2. Select the text cellview and click <i>OK</i>. <p>For details, see <u><i>Descending Using the Descend Command</i></u> in <i>Virtuoso Schematic Editor L User Guide</i>.</p>

The following table describes some methods to create a new text cellview using Virtuoso Text Editor.

Virtuoso Application	Process to Launch Virtuoso Text Editor
Virtuoso Library Manager	<ol style="list-style-type: none">1. Choose <i>File — New — Cell View</i>. The New File form appears.2. Specify the library, cell, and view. Select one of the HDL languages from the <i>Type</i> list.3. Click <i>OK</i>.
Virtuoso CIW	<ol style="list-style-type: none">1. Choose <i>File — New — Cellview</i>. The New File form appears.2. Specify the library, cell, and view. Select one of the HDL languages from the <i>Type</i> list.3. Click <i>OK</i>.
Virtuoso Schematic Editor	<ol style="list-style-type: none">1. Choose <i>File — New</i>. The New File form appears.2. Specify the library, cell, and view. Select one of the HDL languages from the <i>Type</i> list.3. Click <i>OK</i>.

Understanding the Graphical User Interface

The following figure illustrates the main form of Virtuoso Text Editor.



The title bar of Virtuoso Text Editor indicates the library, cell, and text cellview opened in the editor, along with the read or write mode and the language of the text cellview.

Virtuoso Text Editor User Guide

Introduction

The following table describes the key graphical user interface components of Virtuoso Text Editor.

UI Component	Description
Text cellview area	<p>Displays the contents of the text cellview.</p> <p>You can open the cellview in edit or read-only mode. You can open multiple text, schematic, and layout cellviews in different tabs of a single window.</p>
Navigator assistant	<p>Provides facilities to view objects across the design hierarchy using a tree representation.</p> <p>For details, see <i>The Navigator Assistant</i> in <i>Virtuoso Schematic Editor L User Guide</i>. Also see <i>“Navigating a Design Hierarchy Containing Text Cellviews”</i> on page 38.</p>
Menu bar	
<i>Launch</i> menu	<p>Provides the option to launch the NC-Verilog Integration Environment, if a Verilog text cellview is opened.</p> <p>For details on this environment, see <i>Virtuoso NC-Verilog Environment User Guide</i>.</p>

Virtuoso Text Editor User Guide

Introduction

UI Component	Description
--------------	-------------

<i>File menu</i>	<ul style="list-style-type: none">■ <i>New</i> – Create a new cellview.■ <i>Open</i> – Open an existing cellview.■ <i>Close</i> – Close the displayed text cellview.■ <i>Save</i> – Save the text cellview without checking for syntax errors.■ <i>Check</i> – Check the syntax, save, and parse the text cellview.■ <i>Extract</i> – Create the database of instances, nets, and pins in the text cellview after checking, saving, and parsing the cellview. This feature also performs cross-view checks. If the symbol of the text editor does not exist, the feature prompts you to create it.■ <i>Make Editable/Read-Only</i> – Switch text cellview mode to read-only or edit.■ <i>Discard Edits</i> – Discard unsaved edits and reload the text cellview.■ <i>Reload</i> – Reload the text cellview. This option is useful when you edit the cellview in an external editor and want to view the changes in Virtuoso Text Editor.■ <i>Print</i> – Print the contents of the text cellview file.■ <i>Bookmarks</i> – Add and manage bookmarks. <p>For details, see <u>Bookmarks and Views in Virtuoso Studio Design Environment</u>.</p> <ul style="list-style-type: none">■ <i>Cellview list</i> – List the previously opened cellviews for quick access.■ <i>Close All</i> – Close all the tabs in the session window. If there are unsaved changes in the cellview opened in a tab, the application prompts you to save them before closing that tab. If all the tabs are closed, the session window also closes.
------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Virtuoso Text Editor User Guide

Introduction

UI Component	Description
--------------	-------------

<i>Edit menu</i>	■ <i>Undo, Redo</i> – Undo or redo changes.
	■ <i>Copy, Cut, Paste</i> – Copy, cut, and paste content.
	■ <i>Find, Find Previous, Find and Replace</i> – Find and replace text.
	■ <i>Go to Line</i> – Go to a specific line number.
	■ <i>Hierarchy</i> – Navigate the design hierarchy.
	■ <i>Launch External Editor</i> – Open the text cellview in an external editor.
	■ <i>Select All</i> – Select all text in the work area.
<i>View menu</i>	■ <i>Show Line Numbers</i> – Show or hide the line numbers.
	■ <i>Parser Log File</i> – View the text cellview file parser log.
<i>Create menu</i>	■ <i>Cellview From Cellview</i> – Create a new cellview from an existing cellview. For details, see “ Creating a Cellview from an Existing Cellview ” on page 23 and Automatically Creating a Cellview from another Cellview in <i>Virtuoso Schematic Editor L User Guide</i> .
<i>Check menu</i>	■ <i>Pin Order</i> – Check the pin order. This feature is useful for text cellview files where terminals of instances are connected by sequence.
	■ <i>Remove All Markers</i> – Remove all displayed markers, such as syntax error markers.

Virtuoso Text Editor User Guide

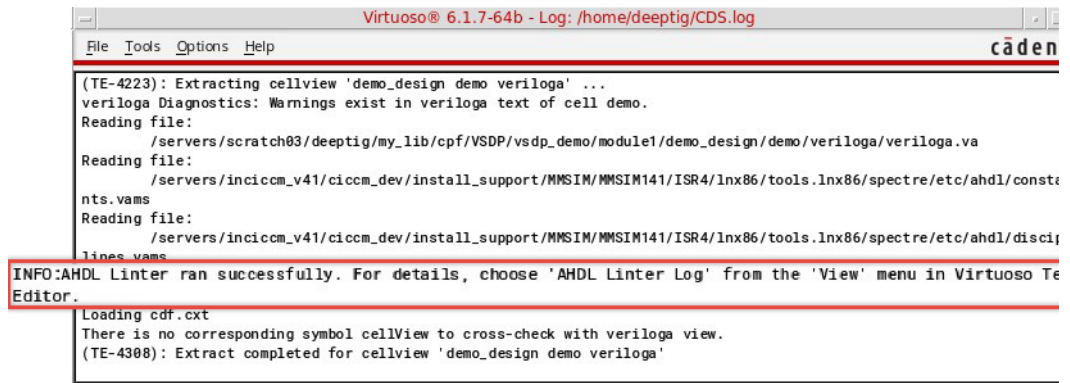
Introduction

UI Component Description

Options menu ■ *Editor* – Enable or disable the default behavior of opening text cellviews in an external editor for editing.

You can set the environment to always open a text cellview for editing in the external editor. To revert this setting, open the text cellview in read-only mode in Virtuoso Text Editor and reset the option.

- *Enable AHDL Linter Static Check*– Enable AHDL Linter static check in Virtuoso Text Editor. You can also enable this check by using “[enableAhdllintStaticCheck](#)” on page 68. When you check the syntax, save the text cellview file, and create the database of instances, nets, and pins in the text cellview, you see following message in CIW.



The screenshot shows the Virtuoso Text Editor window titled "Virtuoso® 6.1.7-64b - Log: /home/deeptig/CDS.log". The menu bar includes "File", "Tools", "Options", and "Help". The "cadence" logo is in the top right corner. The command window displays the following text:

```
(TE-4223): Extracting cellview 'demo_design demo veriloga' ...  
veriloga Diagnostics: Warnings exist in veriloga text of cell demo.  
Reading file:  
          /servers/scratch03/deeptig/my_lib/cpf/VSDP/vsdp_demo/module1/demo_design/demo/veriloga/veriloga.va  
Reading file:  
          /servers/inciccm_v41/ciccm_dev/install_support/MMSIM/MMSIM141/ISR4/lnx86/tools.lnx86/spectre/etc/ahdl/const  
nts.vams  
Reading file:  
          /servers/inciccm_v41/ciccm_dev/install_support/MMSIM/MMSIM141/ISR4/lnx86/tools.lnx86/spectre/etc/ahdl/disci  
lines.vams  
INFO:AHDL Linter ran successfully. For details, choose 'AHDL Linter Log' from the 'View' menu in Virtuoso Te  
Editor.  
Loading cdf.cxt  
There is no corresponding symbol cellView to cross-check with veriloga view.  
(TE-4308): Extract completed for cellview 'demo_design demo veriloga'
```

Virtuoso Text Editor User Guide

Introduction

UI Component	Description
--------------	-------------

<i>Window menu</i>	■ <i>Assistants</i> – Access the Navigator assistant. For details, see “Navigating a Design Hierarchy Containing Text Cellviews” on page 38.
	■ <i>Toolbars</i> – View or hide the toolbars and customize the <i>File</i> and <i>Edit</i> toolbars using Toolbar Manager. For details, see Using and Resetting Toolbar Manager . The available toolbars are <i>File</i> , <i>Edit</i> , <i>Bookmarks</i> , <i>Go</i> , and <i>Workspaces</i> .
	■ <i>Workspaces</i> – Set the workspace. For details, see Virtuoso Workspaces .
	■ <i>Tabs</i> – Navigate between the opened tabs. You can also close the current or other opened tabs.
	■ <i>Copy Window</i> – Copy the current window. Any changes you make in the opened text cellview file is reflected in all the copies of the window.
<i>Help menu</i>	■ Access help and online support.

Toolbars

Note: You can customize the *File* and *Edit* toolbars. For details, see [Using and Resetting Toolbar Manager](#).

<i>File toolbar</i>	■ Create or open a text cellview.
	■ Check the syntax, save the text cellview file, and create the database of instances, nets, and pins in the text cellview.
	■ Save the text cellview file without checking for syntax errors.
<i>Edit toolbar</i>	■ Cut, copy, paste content.
	■ Undo and redo changes.
	■ Search the text cellview content.
<i>Bookmark toolbar</i>	■ Access and manage bookmarks.
	For details, see Bookmarks and Views in Virtuoso Studio Design Environment .

Virtuoso Text Editor User Guide

Introduction

UI Component	Description
--------------	-------------

<i>Go toolbar</i>	■ Navigate the design hierarchy. For details, see <i><u>Navigating Cellviews and Hierarchies</u></i> in <i>Virtuoso Studio Design Environment User Guide</i> .
<i>Workspaces toolbar</i>	■ Set and optimize the workspace. For details, see <i><u>Virtuoso Workspaces</u></i> in <i>Virtuoso Studio Design Environment User Guide</i> .

Note: If you are using a design management system, a menu for the data management operations appears in the menu bar. For details, refer to the documentation of the design management system.

Virtuoso Text Editor User Guide

Introduction

Working With Text Cellviews

Virtuoso Text Editor provides an environment to work with text cellviews. For an overview of this editor, see [Chapter 1, “Introduction.”](#)

This chapter includes the following topics on working with text cellviews:

- [Creating Cellviews](#)
- [Editing Text Cellviews](#)
- [Generating the Databases of Text Cellviews](#)
- [Viewing the File Parse Log](#)
- [Navigating a Design Hierarchy Containing Text Cellviews](#)

Creating Cellviews

Virtuoso Text Editor lets you create new text cellviews. This application also lets you create a cellview using an existing cellview as the source. The source and new cellviews can be of any type, such as text, schematic, or symbol.

In addition to Virtuoso Text Editor, Virtuoso provides various other tools to create digital and analog text cellviews. For example, Virtuoso Verilog In lets you import modules in an external Verilog file as text cellviews in a Virtuoso library. Other such import tools include Virtuoso VHDL Import for VHDL files and Virtuoso Spice In for CDL, HSpice, Spectre, and SPICE netlists. You can continue to use these tools to create text cellviews, while using Virtuoso Text Editor to edit the text cellviews. For details on these import tools, see the following guides:

- [*Verilog In for Virtuoso Design Environment User Guide and Reference*](#)
- [*VHDL In for Virtuoso Design Environment User Guide and Reference*](#)
- [*Design Data Translator's Reference*](#)

You can also use the command `cdsTextTo5x` to import Verilog, SystemVerilog, and VHDL text files into the Virtuoso design environment. For details, see [Importing Design Data by Using cdsTextTo5x](#).

This section describes the creation of cellviews using Virtuoso Text Editor. It includes the following topics:

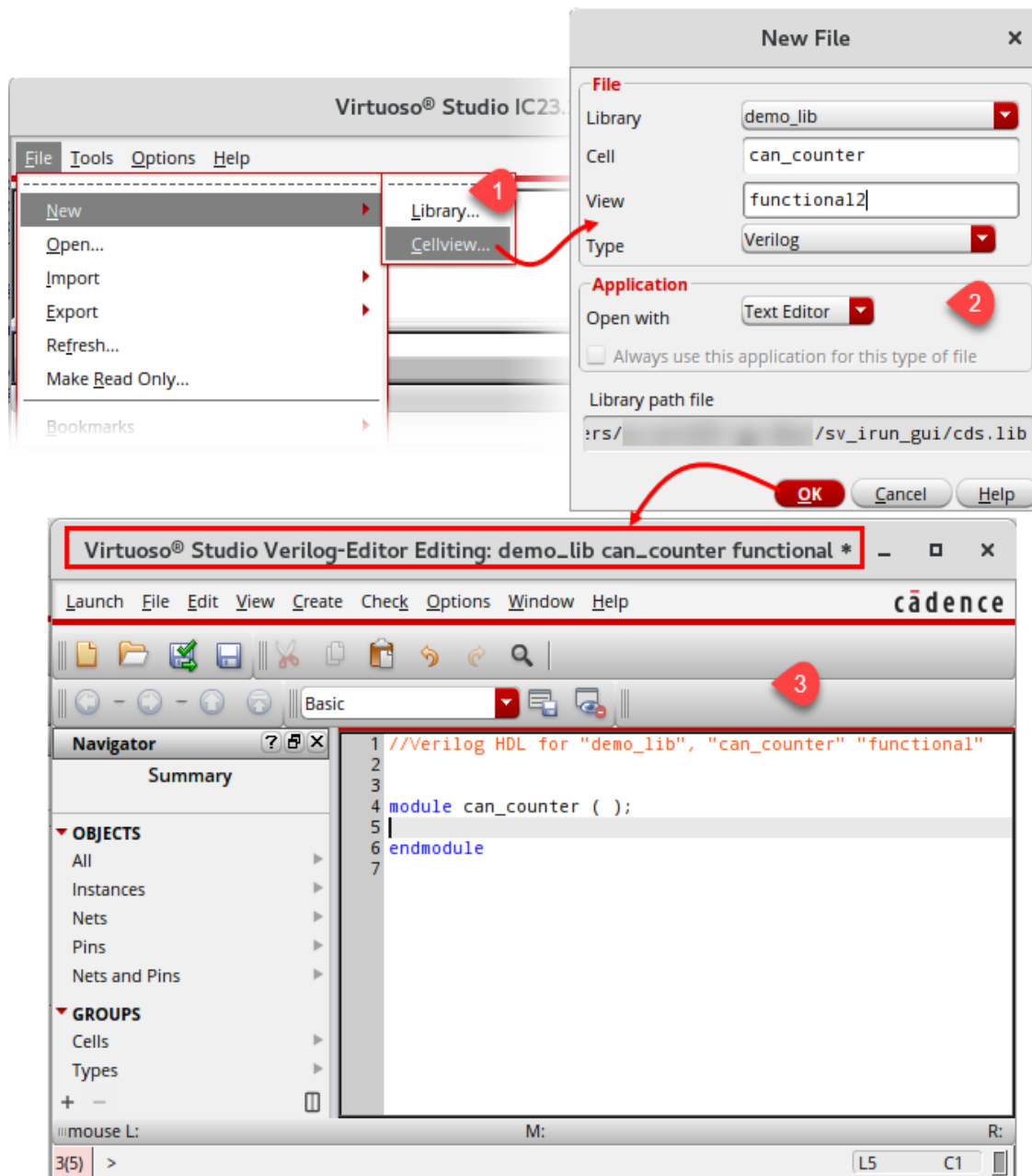
- [Creating a Text Cellview](#)
- [Creating a Cellview from an Existing Cellview](#)

Virtuoso Text Editor User Guide

Working With Text Cellviews

Creating a Text Cellview

There are several ways to initiate the creation of a new text cellview, as described in [Launching Virtuoso Text Editor](#). The following figure illustrates how you initiate the text cellview creation from Virtuoso Library Manager.



Virtuoso Text Editor User Guide

Working With Text Cellviews

To create a text cellview:

1. Open the New File form.

You can open this form from Virtuoso CIW, Virtuoso Library Manager, and other Virtuoso tools. For details, see [Launching Virtuoso Text Editor](#).

To open the New File form from Virtuoso Text Editor, choose *File — New*.

2. Do the following:

- ☐ Specify the library, cell, and view name in their respective fields.
- ☐ Select the HDL from the *Type* drop-down list. For example, to create a Verilog text cellview, select `Verilog`.
- ☐ Set the *Open with* option to *Text Editor*.

3. Click *OK*.

A new blank cellview opens in Virtuoso Text Editor and the directory structure of the cellview is created in the specified library.

Edit and save the text cellview. When you close the new text cellview, Virtuoso Text Editor prompts you to save the symbol of the cellview.

When you close the Virtuoso session without saving the changes in a text cellview, the Save All form displays. You can choose to save or discard the changes in the cellview before Virtuoso closes.

Note: You can set up a config cellview using Virtuoso Schematic Editor or Virtuoso Text Editor. For details, refer to [Setting Up a Config Cellview using Virtuoso Schematic Editor or Virtuoso Text Editor](#).

Important

DSPF, Spectre, and Spice views are available by default. You can create a DSPF view using cellview to cellview functionality so that it can be included in a DSPF file in Spectre and AMS netlisting. Create a blank text cellview and copy DSPF file into the cellview. When using the DSPF view in ADE simulation, the DSPF file is automatically included in the netlist.

Note: Performing *Check and Save* operation on DSPF views may cause significant memory consumption. To avoid this, enable the `subcktHeaderAnalyzer` parser by setting the `cellHeaderAnalyzer.cdsenv` variable to `t`. `subcktHeaderAnalyzer` checks excessive memory consumption by importing only the interface information of DSPF, such as, subckt name and terminals.

Creating a Cellview from an Existing Cellview

Using Virtuoso Text Editor, you can create a cellview using another cellview as a source. For example, you can create a symbol cellview from a Verilog text cellview.

To create a cellview from an existing cellview, you specify the source library, cell, and view and the destination type and view. The destination cellview is created in the same source library and cell. Depending on the destination cellview type, you can provide additional information. For example, to create a symbol cellview from a Verilog cellview, you can choose to specify additional information, such as the left, right, top, and bottom pins.

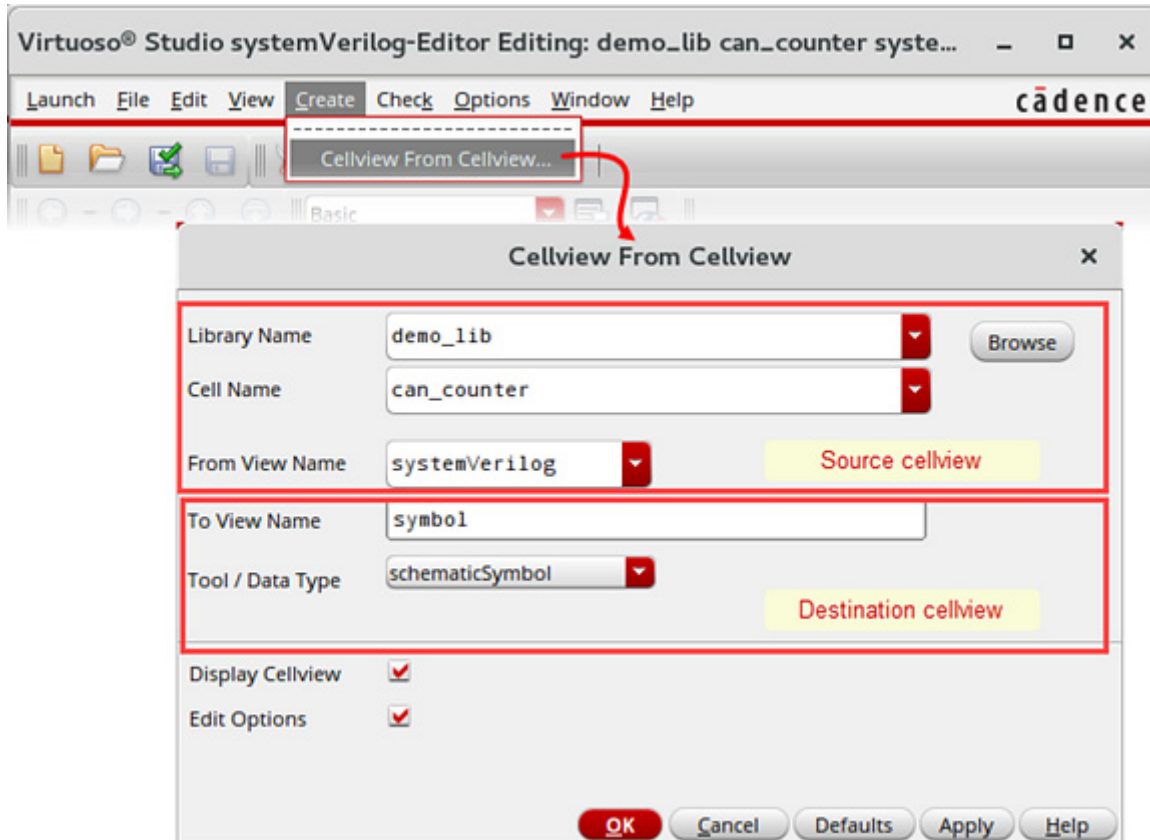
Additional Information

- If the destination cellview already exists, the application prompts you to replace that cellview or cancel the operation. If the symbol of a text cellview does not exist, Virtuoso Text Editor prompts you to generate it when you close that text cellview.
- When creating a cellview from an existing cellview, the `pc.db` files may be overwritten. To prevent these files from being overwritten, set the `vmsUpdatePcdb` flag to `nil`. This flag is set to `t` by default.
- The creation of a cellview from a cellview is useful for various design approaches. For example, you can create a symbol view for a Verilog digital top-level cellview. You can then instantiate this symbol in the top-most schematic design. When you close a text cellview that does not have a corresponding symbol view, Virtuoso Text Editor prompts you to save the symbol of the cellview.
- The feature to create a new cellview from another cellview in Virtuoso Text Editor is similar to the one in Virtuoso Schematic Editor. For details, see [Automatically Creating a Cellview from Another Cellview](#) in *Virtuoso Schematic Editor L User Guide*.

Virtuoso Text Editor User Guide

Working With Text Cellviews

The following figure illustrates how you can create a cellview from a cellview.



To create a cellview from a text cellview:

1. Choose *Create — Cellview From Cellview*.

The Cellview From Cellview form appears.

2. Specify the source and destination cellviews and options.

Note: The library, cell, and view of the currently opened text cellview appear in the respective fields, which you can change. For this, click *Browse* and choose the library, cell, and view from Library Browser.

Field	Description
Source cellview	
<i>Library Name</i>	Specify the library name of the source cellview.
<i>Cell Name</i>	Specify the cell name of the source cellview.

Virtuoso Text Editor User Guide

Working With Text Cellviews

Field	Description
<i>From the View Name</i>	Select the source view name.
Destination cellview	
<i>To View Name</i>	Specify the destination view name.
<i>Tool / Data Type</i>	Select the destination view type. The <i>To View Name</i> field gets updated with the default view name of the selected type.
Options	
<i>Display Cellview</i>	Select to open the new cellview in a new window.
<i>Edit Options</i>	Select to edit any additional options before the cellview creation.

3. Click *OK* or *Apply*.

If you selected *Edit Options* and the destination cellview has additional edit options, the appropriate form with those options appears. For example, if you chose to create a symbol view and have selected *Edit Options*, the Symbol Generation Options form appears when you click *OK*.

If the form for additional edit options appears, specify the required options and click *OK*.

The destination cellview is created using the data in the source cellview.

For VHDL views, you can also create an entity view from the architecture when symbol view is present. For this, set the `vhdlCreateEntityFromArch` SKILL flag to `t`.

Additionally, if *Display Cellview* was selected, the new cellview opens in the appropriate editor. You can also access this cellview from Virtuoso Library Manager.



Tip

You can copy and rename text cells from Virtuoso Library Manager. Virtuoso Library Manager also provides comprehensive features to copy data. For details, see [Working with Text Cellviews](#) and [Copying Data](#) in *Cadence Library Manager User Guide*.

Editing Text Cellviews

This section described the following topics:

- [Switching Between Edit Mode and Read-Only Mode](#)
- [Checking Syntax](#)
- [Showing Line Numbers and Going to a Line Number](#)
- [Editing a Text Cellview File in an External Editor](#)
- [Checking Pins](#)
- [Using Other Editing Features](#)

Switching Between Edit Mode and Read-Only Mode

You can edit a text cellview opened in Virtuoso Text Editor in edit mode. You can switch the text cellview mode between read-only and edit. The default background color of the content indicates the mode, as described in the following table.

Default Background Color	Description
White	The text cellview is opened in edit mode.
Gray	The text cellview is opened in read-only mode.

You can change the default background color using the environment variables `inScopeReadBGColor` and `inScopeWriteBGColor`. For details, see [Appendix B, “Environment Variables.”](#)

To switch between read-only and edit modes:

- ➔ Choose *File — Make Editable*, or *File — Read-Only*. The available option depends on the current mode.

Checking Syntax

After editing a text cellview, ensure that there are no syntax errors. If the text cellview has syntax errors, operations like file parsing and database generation fail.

Virtuoso Text Editor lets you check syntax errors in the text cellview opened in edit or read-only mode. The editor highlights the syntax errors. When you place the cursor over a syntax error, the error description appears as a tool tip.

The following figure provides an example of checking a text cellview for syntax errors.

1. Syntax error in the text file.

```
module can counter ( clk, load, count, dispense, empty );
input clk, load, dispense;
input [7:0] count;
output empty;
&&&&
reg[7:0] left;
wire empty = |left;
always @(negedge clk)
    if (load && !dispense)
        left <= count;
    else if (!load && dispense)
        left <= left -1;
endmodule
```

2. Check the syntax.

marker message:
Error: expecting the keyword 'endmodule' [12.1(IEEE)].

```
module can counter ( clk, load, count, dispense, empty );
input clk, load, dispense;
input [7:0] count;
output empty;
&&&&
reg[7:0] left;
wire empty = |left;
always @(negedge clk)
    if (load && !dispense)
        left <= count;
    else if (!load && dispense)
        left <= left -1;
endmodule
```

3. Remove the error. Then check the syntax.

```
module can counter ( clk, load, count, dispense, empty );
input clk, load, dispense;
input [7:0] count;
output empty;
reg[7:0] left;
wire empty = |left;
always @(negedge clk)
    if (load and !dispense)
        left <= count;
    else if (!load and dispense)
        left <= left -1;
endmodule
```

To check for syntax errors in the text cellview opened in Virtuoso Text Editor:

➡ Choose *File — Check*.

The syntax errors get highlighted.



Tip

- Virtuoso Text Editor also checks and saves the text cellview when you generate the database of that cellview. See [“Generating the Databases of Text Cellviews”](#) on page 34.
- You can remove the syntax error markers by choosing *Check — Remove All Markers*.
- For further information on the syntax errors in the text cellview file, view the parser log. To access this log, choose *View — Parser Log File*.
- To save the text cellview file without checking the syntax, choose *File — Save*. This option is useful when you want to save an intermediate version of the file that you plan to update later.

Showing Line Numbers and Going to a Line Number

You can view the line numbers of the text cellview file opened in Virtuoso Text Editor. You can also go to a specific line number. This feature is useful if the file is large and you want to go to a specific line number.

To show or hide the line numbers:

➡ Choose *View — Show Line Numbers*.

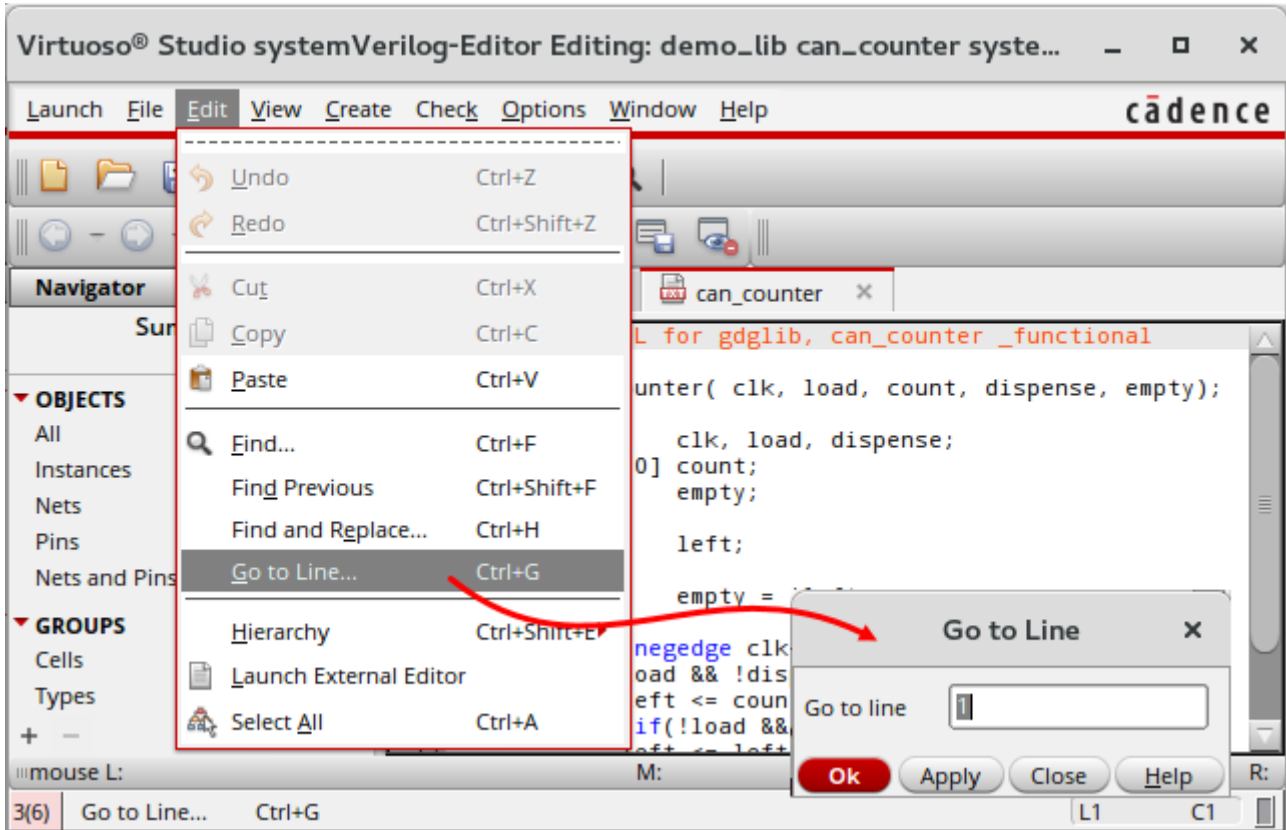
If the line numbers are hidden, they become visible.

You can configure Virtuoso Text Editor to display the line number by default using the `showLineNumbersInSideBar` environment variable. For details, see [showLineNumbersInSideBar](#) on page 77.

Virtuoso Text Editor User Guide

Working With Text Cellviews

The following figure illustrates how you go to a line number.



To go to a line number:

1. Choose *File* — *Go to Line*.

The Go to Line form appears.

2. Type the line number.
3. Click *OK* or *Apply*.

The cursor is placed in the beginning of the specified line number and that line is highlighted.

Notes:

- You can go to a specific line number, even if the line numbers are hidden.
- The bottom-right corner of the Virtuoso Text Editor window displays the current line number and character position. For example, if you place the cursor in the beginning of the text cellview, the bottom-right corner displays L1 C1.

- You can retrieve and set the character position of the cursor in the text cellview using the APIs [teGetCursorPosition](#) and [teSetCursor](#).

Editing a Text Cellview File in an External Editor

From Virtuoso Text Editor, you can open a text cellview file in an external editor, like the vi editor. You can set the external editor through the shell variable `EDITOR` or the SKILL variable `editor`.

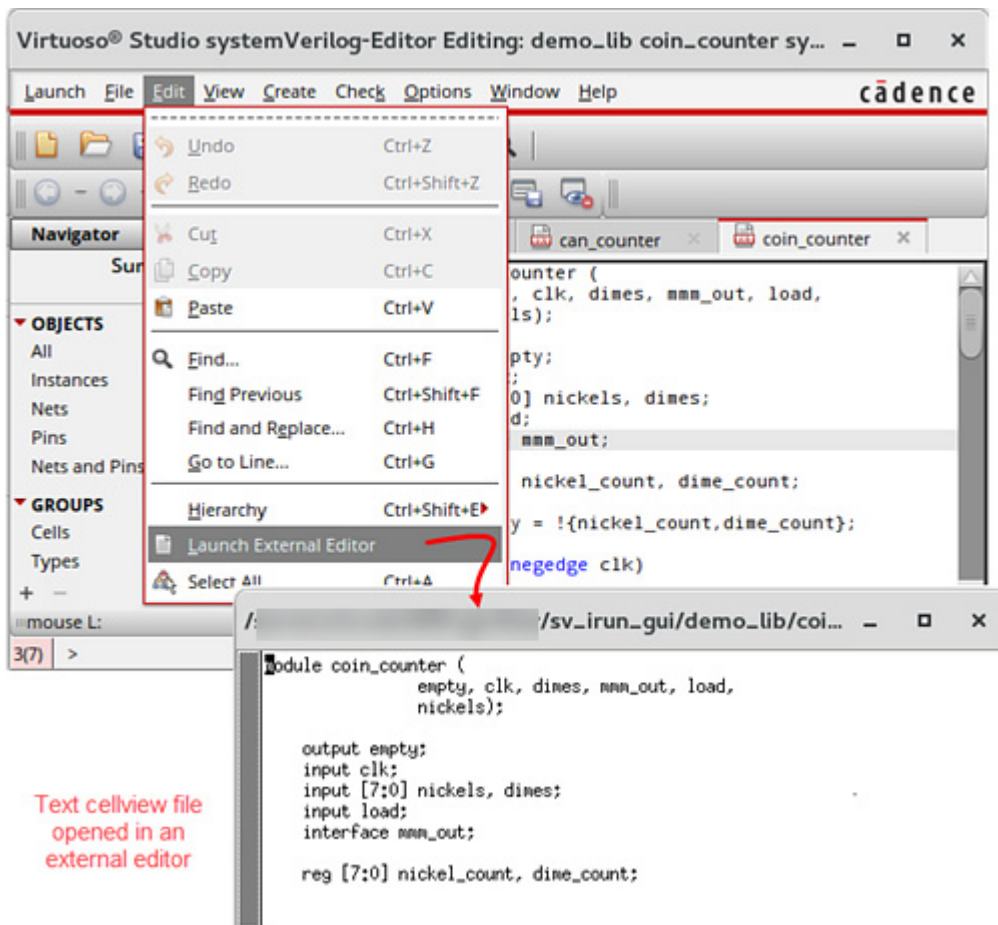
When you open the file in the external editor, that file becomes read-only in Virtuoso Text Editor. After you edit and close the file in the external editor, Virtuoso Text Editor prompts you to reload the file. You can then edit that file in Virtuoso Text Editor.

When you edit, save, and close the file in the external editor, the file is parsed. You can view the parser messages in Virtuoso CIW. If the parsing fails, a message appears, prompting you to view the parse error log and correct the errors.

Virtuoso Text Editor User Guide

Working With Text Cellviews

The following figure illustrates how you open a text cellview file in the external editor.



To open the displayed text cellview file in an external editor:

- ➡ Choose *Edit — Launch External Editor*.

The file becomes read-only in Virtuoso Text Editor and opens in the external editor.

Notes:

- If you choose to open a read-only file in the external text editor, the application confirms the action.
- If the file has unsaved changes, the application prompts you to save them before opening the file in the external editor.

You can configure Virtuoso to always open text cellviews in the external editor for editing. For this, use the [useExternalEditor](#) environment variable, or the [Text Editor Options Form](#). To always open text cellviews in an external editor, select the option, select the option *Always*

Virtuoso Text Editor User Guide

Working With Text Cellviews

Edit in External Editor in the Text Editor Options Form and set the SKILL variable `hdlReadOnlyModeEditorCommand` to the external editor or viewer.

For details on the SKILL variables `hdlReadOnlyModeEditorCommand` and `editor`, see “Specifying an Editor for Text Files” in *Virtuoso NC-Verilog Environment User Guide*.

Checking Pins

You can check the name, order, and number of ports in the different views of the current cell. This feature is useful for text cellview files that connect terminals of instances sequentially. Virtuoso Text Editor also checks the ports when it extracts the database of the text cellview.

To check the ports in all the views parallel to the current text cellview:

➔ Choose *Check – Pin Order*.

The results of the port checks appear in Virtuoso CIW. The application checks ports in the different views of the cell. If any discrepancies are found, the Port Mismatch form appears. This form lists the views with mismatching ports, along with the issues and recommended corrective actions.

You can configure the behavior of the port check functionality using the environment variables `disablePortOrderPopup` and `disablePortOrderCheck`. For details, see Resolving Pin Mismatch.

If the current cell does not contain a symbol view, running the *Check and Save* command creates the symbol view automatically. Once the symbol view has been created in the current cell, choose *Check – Cellview From Cellview* to either regenerate the symbol view or generate a new symbol view.

By default, the text-to-symbol generator sorts the pins for this symbol view in alphanumeric order. You can change the order in which pins for the symbol are displayed in the schematic by using the environment variable `ssgSortPins`.

For example, consider a text cellview with the following module definition:

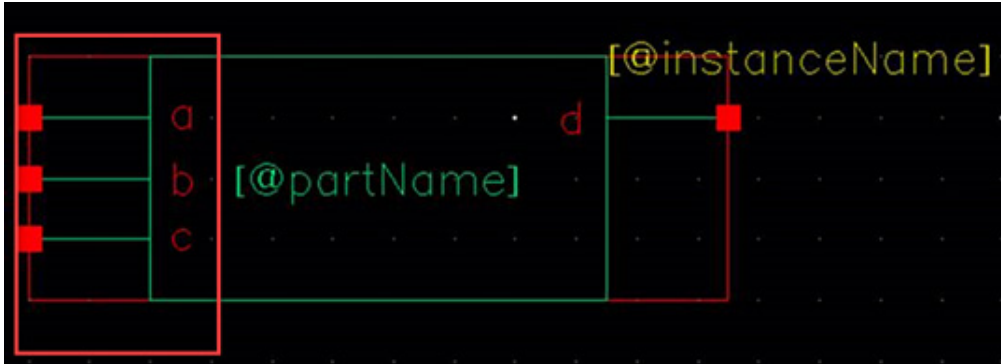
```
module cellA ( a , c , b , d )
    input a;
    input c;
    input b;
    output d;
endmodule
```

If a symbol view does not exist in the current cell, `cellA`, run the *Check and Save* command, otherwise, choose *Create – CellView to CellView*. Double-click the symbol view

Virtuoso Text Editor User Guide

Working With Text Cellviews

in the Library Manager window to view the symbol in the Virtuoso Symbol Editor. Observe that the pins are displayed in the default order where pins are sorted by their names.



Check the current value of the `ssgSortPins` environment variable by running the following command in the Virtuoso CIW:

```
envGetVal("schematic" "ssgSortPins")
```

It is set to `alphanumeric`.

Now, set the environment variable `ssgSortPins` as follows in the CIW and regenerate the symbol by choosing *Create – CellView to CellView*:

```
envSetVal("schematic" "ssgSortPins" 'cyclic "geometric")
```

The Virtuoso Symbol Editor then displays the symbol with the pins sorted in the same order as the module definition.



Using Other Editing Features

You can use the following features to edit the text cellview opened in Virtuoso Text Editor:

- Find and replace strings.
- Cut, copy, and paste content.

■ Undo and redo changes.

Use the *Edit* menu and *Edit* toolbar to perform these edit operations. For details on the menu bar and toolbars, see [“Understanding the Graphical User Interface”](#) on page 11.

Note: You cannot undo operations such as text cellview database generation. Additionally, you cannot undo changes performed before database generation.



Virtuoso Text Editor supports copying and pasting content up to a maximum limit of 8KB, which is equivalent to 8192 characters. An error message is displayed when this limit is exceeded.

Generating the Databases of Text Cellviews

Virtuoso Text Editor creates the database of instances, nets, and pins in a text cellview when you save and close the text cellview file. You can also manually create or update the database of a text cellview opened in edit or read-only mode.

Note: Virtuoso Text Editor does not provide the option to extract the text cellview databases from Spectre, SPICE, HSPICE, and CDL files.

Before generating the database of a text cellview, Virtuoso Text Editor checks the syntax in the file. To generate the database of a text cellview, Virtuoso Text Editor uses the appropriate parser, depending on the cellview HDL file. For example, to parse a Verilog, Verilog-AMS, or SystemVerilog file, the editor uses the Native Code Verilog compiler (ncvlog). For a VHDL file, the editor uses the Native Code VHDL compiler (ncvhdl). Similarly, the editor uses different parsers for different types of HDLs. The parser checks the syntax, design, connectivity, and masters. If the checks pass, it creates and stores the database information in the `netlist.oa` and `data.dm` files within the directory structure of the text cellview.

If the application encounters file parse errors, the database is not generated. In this case, you can view the log of the file parse operation to investigate the errors. For details, see [“Viewing the File Parse Log”](#) on page 36.

It is possible that some instances in a text cellview are not bound. In such cases, Virtuoso Text Editor indicates the issues in Virtuoso CIW. The following report is an example of how

Virtuoso Text Editor User Guide

Working With Text Cellviews

instance binding information appears in Virtuoso CIW. The first four instances mentioned in the report are not bound to any cellview.

Instance binding report.

master name	lib	cell	view
PLL_160MHZ_MDIV	nil	nil	nil
PLL_160MHZ_PDIV	nil	nil	nil
gpd090_nmoscap2v	nil	nil	nil
or2_4x_hv	nil	nil	nil
PLL_ARST_DIG	amsPLL	PLL_ARST_DIG	symbol
PLL_ARST	amsPLL	PLL_ARST	symbol
PLL_VCO_320MHZ	amsPLL	PLL_VCO_320MHZ	symbol
PLL_PFD	amsPLL	PLL_PFD	symbol
PLL_CP	amsPLL	PLL_CP	symbol
PLL_160MHZ_LF	amsPLL	PLL_160MHZ_LF	symbol

If there are unbound instances and the design flow requires you to correct such issues, perform corrective actions and regenerate the database. For example, if the design flow requires netlist generation, the unbound instance must be corrected to avoid issues.

After generating the database, Virtuoso Text Editor checks for any mismatch in the ports across different views of the cell.

Note: You can configure the behavior of the port check functionality. For details, see [*Resolving Pin Mismatch*](#) in *Virtuoso Schematic Editor L User Guide*.

To generate the database of the text cellview opened in Virtuoso Text Editor manually, do one of the following:

- ➔ Click the *Build a database* button on the toolbar.
- ➔ Choose *File — Extract*.

If you edit the text cellview and close it without regenerating the database, the application checks and parses the file, and generates the database automatically.

Viewing the File Parse Log

Virtuoso Text Editor uses the appropriate parser to parse the file of a text cellview to perform operations, such as generating the cellview database. When the parser processes a file, it maintains a log. You can view this log for details. For example, you can view the log to investigate parse errors.

In addition to information about any errors, the parse log of a text cellview file provides additional information, such as:

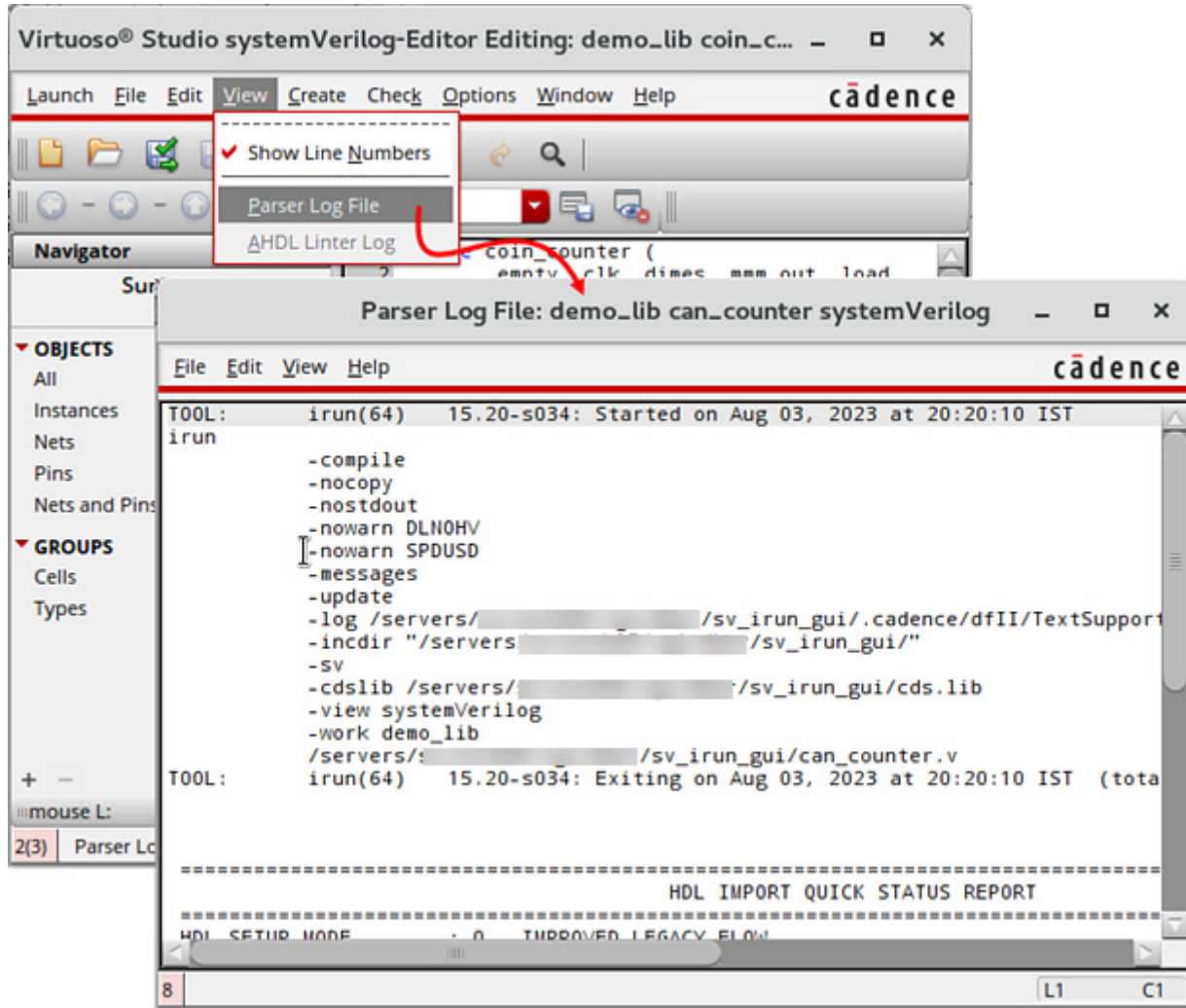
- The parser name and version.
- The name and path of the file that was parsed.
- The start time and the end time of the file parsing operation.
- The command-line options used for parsing the file.

Note: Verilog-A files are parsed using the Spectre binary. The parse log of a Verilog-A file does not include additional information.

Virtuoso Text Editor User Guide

Working With Text Cellviews

The following figure illustrates how you can view the log of a file parsing operation in Virtuoso Text Editor.



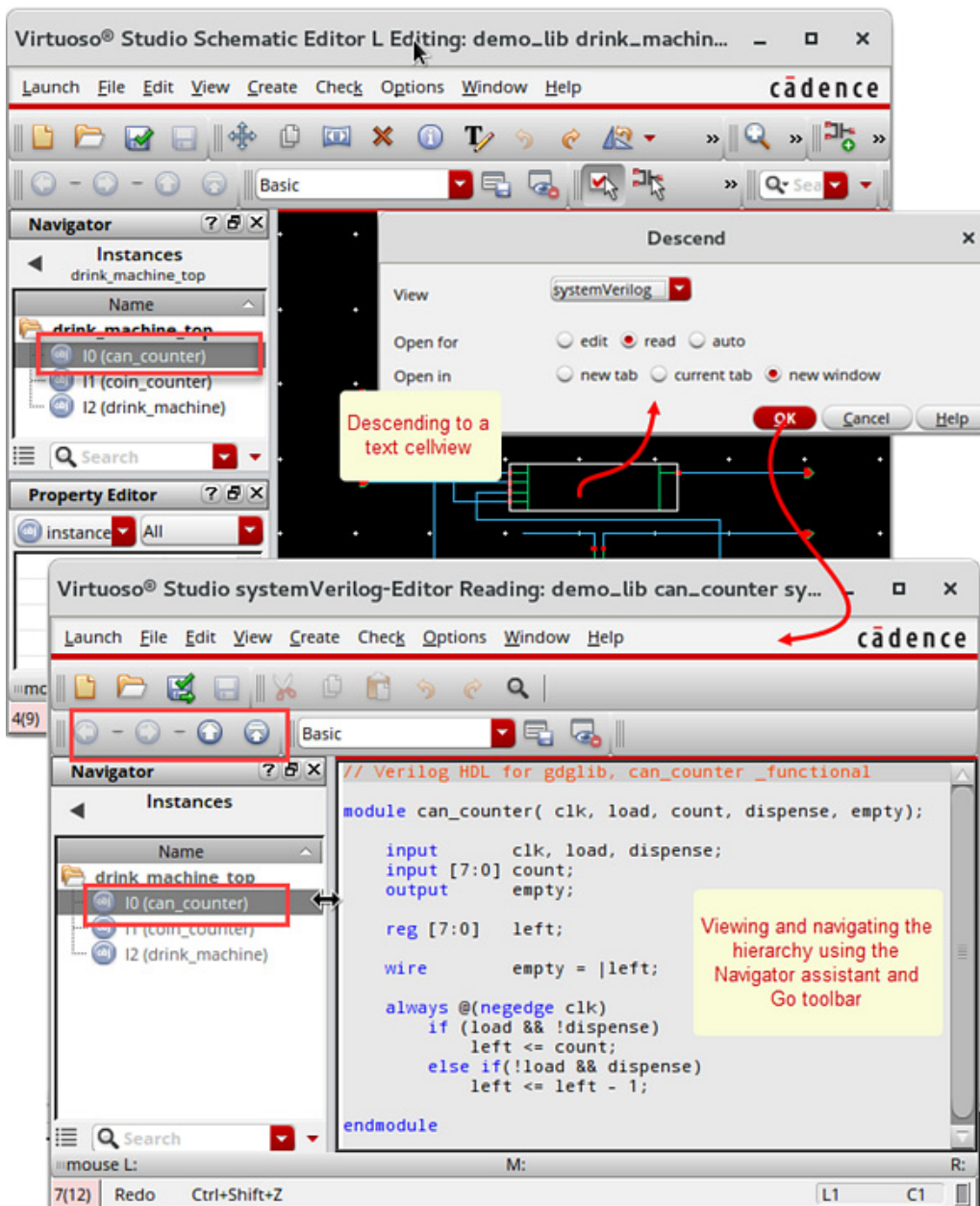
To view the file parse log of the currently opened text cellview:

➔ Choose *View — Parser Log File*.

The Parse Log File window appears with the log.

Navigating a Design Hierarchy Containing Text Cellviews

You can instantiate a symbol of a text cellview in a schematic view. You can then descend to this text cellview from the schematic view. The following figure illustrates how you navigate a design containing text cellviews. In this example of the `drink_machine` design, the `drink_machine_top` schematic view contains the symbol of the SystemVerilog text cellview `can_counter`. The figure illustrates how you can descend to `can_counter` instance `I0` from `drink_machine_top`.



Virtuoso Text Editor User Guide

Working With Text Cellviews

To descend to a text cellview from a schematic:

1. Select the text cellview symbol instantiated in the schematic view.
2. Do one of the following to open the Descend form:
 - ☐ Press **E**.
 - ☐ Double-click the symbol.
 - ☐ Choose *Edit — Hierarchy — Descend Edit*, or *Descend Read*.
3. Specify the mode and location to open the text cellview.
4. Press **OK**.

The text cellview opens. You can open multiple text, schematic, and layout views in different tabs of the same window. You can also open them in the same tab or in a different window.

The Navigator assistant shows the design hierarchy, including the text cellviews.

To access the Navigator assistant, do one of the following:

- ➔ Choose *Windows — Assistants — Navigator*.
- ➔ Right-click the menu bar and select *Assistants — Navigator*.



Tip

To view or hide the Navigator assistant, click the *Toggle Assistants Visibility* button on the *Workspaces* menu.

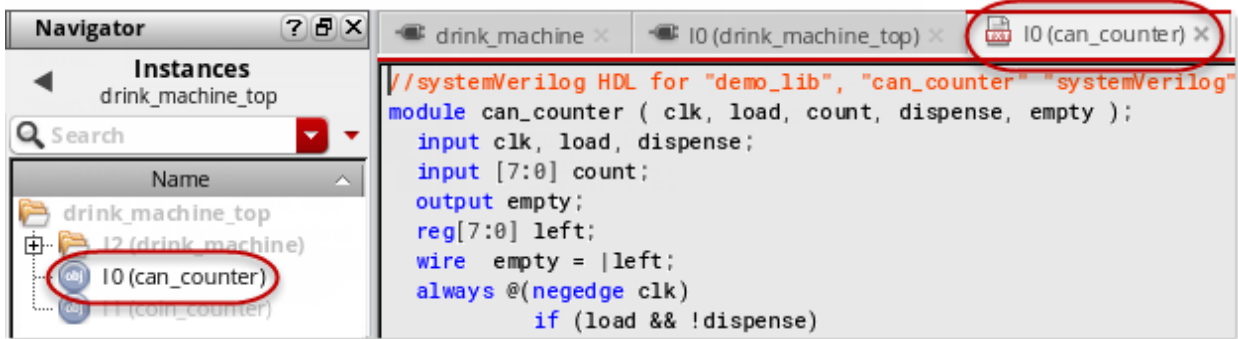
The Navigator assistant is available in Virtuoso Text Editor, Virtuoso Schematic Editor L and XL, and Virtuoso Layout Suite L. For details, see [*The Navigator Assistant*](#) in *Virtuoso Schematic Editor L User Guide*.

Virtuoso Text Editor User Guide

Working With Text Cellviews

Note the following:

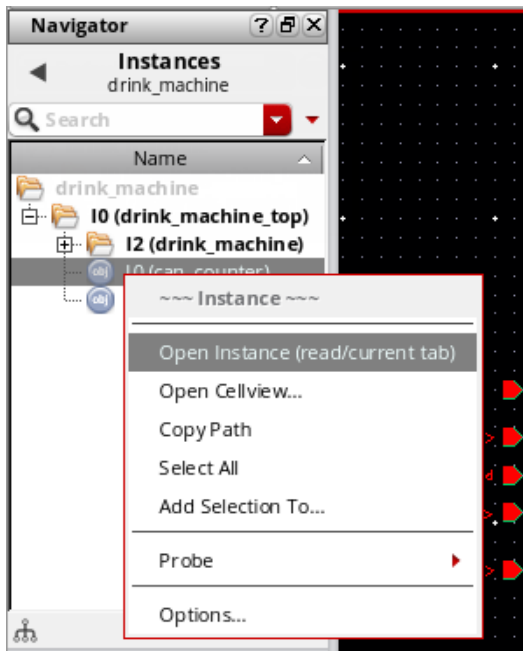
- You can view the design hierarchy in the Navigator assistant. The highlighted entry in the assistant indicates the currently displayed view. The other entries appear in gray text.



- To navigate the design, use any of the following interfaces:
 - The *Go* toolbar
 - The *View — Hierarchy* options
 - The Navigator assistant
- To navigate a design using the Navigator assistant:
 - a. Right-click an instance to view the context menu.
 - b. Select *Open Instance* to open the instance in the design context. The default view of the instance opens in the same tab in read-only mode.

Virtuoso Text Editor User Guide

Working With Text Cellviews



Note: To open the cellview out of the design context, select *Open Cellview*. You cannot navigate the design from the cellview using the *Go* toolbar or *View — Hierarchy* menu options.

Ignoring Explicitly Inherited Terminals for Text Cellviews

The Cross-View Checker form lets you ignore explicitly inherited terminals for text cellviews, such as `functional`, `systemverilog`, `verilogams`, or `symbol` cellviews.

For example, consider that you have a cell `cellA` and it has three views, `schematic`, `symbol`, and `SystemVerilog`.

Here,

- The `symbol` view has one port `a`.
- The `schematic` view has two ports `a` and `b`, where `b` is an explicitly inherited terminal.
- The `systemverilog` view has two ports `a` and `b`.

To ignore inherited terminals:

1. From the schematic window, open the Cross-View checker form.
2. Select *Match Inherited Terminals*.

Virtuoso Text Editor User Guide

Working With Text Cellviews

3. Select *Ignore Terminals with Net Expression and no Terminals in the Other View*.
4. Specify a text view type in the *View Names* field. For example, specify the `functional` view.
5. Select the *Terminal Names* option and specify the name of the terminal in the adjacent field.
6. Click *OK*.
7. Open the view in the Text Editor window and click *Check and Save*.

The inherited terminals are matched while the explicitly inherited terminals with pin mismatches are ignored for the specified cellviews.

Virtuoso Text Editor and HDL Package Setup Functions

This topic lists the Cadence® SKILL functions associated with Virtuoso® Text Editor and Virtuoso HDL Package Setup.

Only the functions listed below 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.

- Text Editor Functions - This following SKILL functions let you perform various tasks using the Virtuoso Text Editor.

[teDiscardEdits](#)

[teGetCursorPosition](#)

[teIsModified](#)

[teSave](#)

[teSaveWithDerivedData](#)

[teSetCursor](#)

[teSetEditMode](#)

[teRegPostExtractTrigger](#)

[teRegPreExtractTrigger](#)

[teUnRegPostExtractTrigger](#)

[teUnRegPreExtractTrigger](#)

[hdlGenerateTextDatabase](#)

- Virtuoso HDL Package Setup Functions - This following SKILL functions let you perform various tasks using the Virtuoso HDL Package Setup.

[hdlPkgCreateEnvSetupFile](#)

[hdlPkgExportXrunArgs](#)

Related Topics

[Working With Text Cellviews](#)

[HDL Package Setup](#)

hdlPkgCreateEnvSetupFile

```
hdlPkgCreateEnvSetupFile(  
    t_fileName  
    [ ?formatLList { t | nil } ]  
)  
=> t / nil
```

Description

Saves the environment variables used by the HDL Package Setup from the current Virtuoso setup database into a specified file. Each name-value pair for a variable is printed in the following format:

```
envSetVal(t_tool[.Partition] t_varName s_varType g_newValue)
```

Arguments

<i>t_fileName</i>	The name of the file in which the settings are saved. You can use shell environment variables to specify the file name. If the specified file exists, it is overwritten. Otherwise, a new file is created.
-------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

?formatLList { t | nil }

Specifies whether long lists must be formatted. If set to *t*, each sublist is printed in a new line for the values of the environment variables, `hdlPkgMakeLibTree` and `hdlPkgXrunArgsFileTableList`.

Valid values are *t* and *nil*. The default is *t*.

The default value *t* makes the file more readable but formatting of the long-list string can take additional time. For faster performance, set *?formatLList* to *nil*.

Value Returned

<i>t</i>	The environment variables used by the HDL Package Setup are saved in a file.
<i>nil</i>	An error was reported.

Virtuoso Text Editor User Guide

Virtuoso Text Editor and HDL Package Setup Functions

Examples

Before making a call to the function, ensure the following:

- The shell environment variable PROJ is set before launching Virtuoso.
- The required settings are applied in the Virtuoso HDL Package Setup form.

The following example describes how to use the `hdlPkgCreateEnvSetupFile` function in the CIW to save the HDL Package Setup settings from the current Virtuoso session.

```
hdlPkgCreateEnvSetupFile("$PROJ/myHDLpkgSetup")
=> t
```

Contents of the file "\$PROJ/myHDLpkgSetup", when `?formatLList` is set to `t`.

```
/*
*****
Virtuoso HDL Package Setup
*****
*/

envSetVal("hdlPkg" "hdlPkgEnableHdlSetup" 'toggle '(t))
envSetVal("hdlPkg" "hdlPkgProjDir" 'string "$PROJ/compileScratch")
envSetVal("hdlPkg" "hdlPkgEnableUVM" 'toggle '(t))
envSetVal("hdlPkg" "hdlPkgUVMVersion" 'string "IEEE")
envSetVal("hdlPkg" "hdlPkgEnableUPF" 'toggle '(t))
envSetVal("hdlPkg" "hdlPkgUseHdlvar" 'toggle '(nil))
envSetVal("hdlPkg" "hdlPkgHdlvarFile" 'string "")
envSetVal("hdlPkg" "hdlPkgAdvancedMode" 'toggle '(t))
envSetVal("hdlPkg" "hdlPkgUseXmlbdirname" 'toggle '(t))
envSetVal("hdlPkg" "hdlPkgXmlbdirname" 'string "XM2109")
envSetVal("hdlPkg" "hdlPkgMakeLibTree" 'string "(
(t \"pkgLib\" \"$PROJ/pkg_source/bill_pkg.sv\" \"\")
(t \"pkgLib\" \"$PROJ/pkg_source/fred_pkg.sv\" \"\")
)")
envSetVal("hdlPkg" "hdlPkgXrunArgsFileTableList" 'string "(
(t \"$PROJ/user_1.f\")
(nil \"$PROJ/user_2.f\")
)")
```

Contents of the file "\$PROJ/myHDLpkgSetup", when `?formatLList` is set to `nil`.

```
/*
*****
Virtuoso HDL Package Setup
*****
*/

envSetVal("hdlPkg" "hdlPkgEnableHdlSetup" 'toggle '(t))
envSetVal("hdlPkg" "hdlPkgProjDir" 'string "$PROJ/compileScratch")
envSetVal("hdlPkg" "hdlPkgEnableUVM" 'toggle '(t))
envSetVal("hdlPkg" "hdlPkgUVMVersion" 'string "IEEE")
envSetVal("hdlPkg" "hdlPkgEnableUPF" 'toggle '(t))
envSetVal("hdlPkg" "hdlPkgUseHdlvar" 'toggle '(nil))
envSetVal("hdlPkg" "hdlPkgHdlvarFile" 'string "")
envSetVal("hdlPkg" "hdlPkgAdvancedMode" 'toggle '(t))
envSetVal("hdlPkg" "hdlPkgUseXmlbdirname" 'toggle '(t))
envSetVal("hdlPkg" "hdlPkgXmlbdirname" 'string "XM2109")
envSetVal("hdlPkg" "hdlPkgMakeLibTree" 'string "((t \"pkgLib\" \"$PROJ/pkg_source/
bill_pkg.sv\" \"\") (t \"pkgLib\" \"$PROJ/pkg_source/fred_pkg.sv\" \"\"))")
```

Virtuoso Text Editor User Guide

Virtuoso Text Editor and HDL Package Setup Functions

```
envSetVal("hdlPkg" "hdlPkgXrunArgsFileTableList" 'string "((t \"$PROJ/  
user_1.f\") (nil \"$PROJ/user_2.f\"))")
```

Related Topics

[hdlPkgExportXrunArgs](#)

hdlPkgExportXrunArgs

```
hdlPkgExportXrunArgs(  
    t_fileName  
    [ ?format { t | nil } ]  
    [ ?isExpandPath { t | nil } ]  
    [ ?userSpecOnly { t | nil } ]  
)  
=> t / nil
```

Description

Exports the `xrun` arguments corresponding to environment variables used by the HDL Package Setup from the current Virtuoso setup database into the specified file.

Arguments

<code>t_fileName</code>	The name of the file in which the <code>xrun</code> arguments are saved. You can use shell environment variables to specify the file name. If the specified file exists, it is overwritten. Otherwise, a new file is created.
-------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<code>?format { t nil }</code>	Specifies that each <code>xrun</code> argument must be printed in a new line. Valid values are <code>t</code> and <code>nil</code> . The default is <code>t</code> .
----------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<code>?isExpandPath { t nil }</code>	Specifies that the specified relative file path or the file path containing shell environment variables must be expanded. Valid values are <code>t</code> and <code>nil</code> . The default is <code>nil</code> .
----------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

`?userSpecOnly { t | nil }`

Specifies that only the user-specified options are exported.

When `?userSpecOnly` is set to `t`, the `-xmlibdirpath`, `-cdslib` *fileName* options are ignored in all cases.

The settings specified for `?userSpecOnly` and the *hdl.var file* check box in the Virtuoso HDL Package Setup form determine whether the user-specified `-hdlvar` option is ignored or considered during export.

When `?userSpecOnly` is set to `t`:

- If the *hdl.var file* check box is not enabled, a dummy `hdl.var` file is automatically generated, but ignored during export because it is not user-specified.
- If the *hdl.var file* check box is enabled and a valid `hdl.var` file is specified, no dummy file is generated. The user-specified `hdl.var` file is exported.

When `?userSpecOnly` is set to `nil`:

- If the *hdl.var file* check box is not enabled, a dummy `hdl.var` is generated and exported.
- If the *hdl.var file* check box is enabled, no dummy file is generated. The user-specified `hdl.var` file is exported irrespective of the value set for `?userSpecOnly`.

Value Returned

<code>t</code>	The environment variables used by the HDL Package Setup are saved in a file.
<code>nil</code>	An error was reported.

Examples

Before making a call to the function, ensure the following:

- The shell environment variable `PROJ` is set before launching Virtuoso.
- The required settings are applied in the Virtuoso HDL Package Setup form.

Virtuoso Text Editor User Guide

Virtuoso Text Editor and HDL Package Setup Functions

The following example describes how to use the `hdlPkgExportXrunArgs` function in the CIW to export the xrun arguments corresponding to the HDL Package Setup settings from the current Virtuoso session into a file.

```
hdlPkgExportXrunArgs("$PROJ/xrunArgs_exported")
=> t
```

Contents of the file `$PROJ/xrunArgs_exported` when `?userSpecOnly` is set to `t`.

```
// Virtuoso HDL Package Setup
// User Name: xxxxxxxx
// Save Time: Mar 16 02:35:52 2022
// Xcelium Version: 22.03-a001-20220315
// IC subversion: ICADVM20.1-64b.XXXX.xxx

// -xmlbdirpath $PROJ/compileScratch
-xmlbdirname XM2109
// -cdslib $PROJ/compileScratch/cds.lib
// -hdlvar $PROJ/compileScratch/hdl.var
-uvm
-uvmhome CDNS-IEEE
-makelib worklib
-sv
${_XRUNROOT_}/inca/files/1801/upf_package.sv
-endlib
-f $PROJ/user_1.f
-makelib pkgLib
$PROJ/pkg_source/bill_pkg.sv
-endlib
-makelib pkgLib
$PROJ/pkg_source/fred_pkg.sv
-endlib
```

Contents of the file `$PROJ/xrunArgs_exported` when `?userSpecOnly` is set to `nil`.

```
// Virtuoso HDL Package Setup
// User Name: xxxxxxxx
// Save Time: Mar 16 02:35:52 2022
// Xcelium Version: 22.03-a001-20220315
// IC subversion: ICADVM20.1-64b.XXXX.xxx

-xmlbdirpath $PROJ/compileScratch
-xmlbdirname XM2109
-cdslib $PROJ/compileScratch/cds.lib
-hdlvar $PROJ/compileScratch/hdl.var
-uvm
-uvmhome CDNS-IEEE
-makelib worklib
-sv
${_XRUNROOT_}/inca/files/1801/upf_package.sv
-endlib
-f $PROJ/user_1.f
-makelib pkgLib
$PROJ/pkg_source/bill_pkg.sv
-endlib
-makelib pkgLib
$PROJ/pkg_source/fred_pkg.sv
-endlib
```

Virtuoso Text Editor User Guide

Virtuoso Text Editor and HDL Package Setup Functions

Related Topics

[hdlPkgCreateEnvSetupFile](#)

teDiscardEdits

```
teDiscardEdits(  
    d_cvId  
)  
=> t / nil
```

Description

Rolls back the recent updates being done in all the opened editors displaying the specified cellview to the last saved changes.

Arguments

<i>d_cvId</i>	Cellview ID of the schematic in which the edits have been discarded.
---------------	----------------------------------------------------------------------

Value Returned

t	Successfully reverted edits in the cellview.
nil	The changes in the cellview could not be reverted.

Example

Consider that the `myCellView` is the id of text cellview opened in Virtuoso Text Editor. You can discard the edits as shown here:

```
myCellView = ddGetObj("myLib" "myCell" "myVerilogView")  
teDiscardEdits(myCellView)
```

Virtuoso Text Editor User Guide

Virtuoso Text Editor and HDL Package Setup Functions

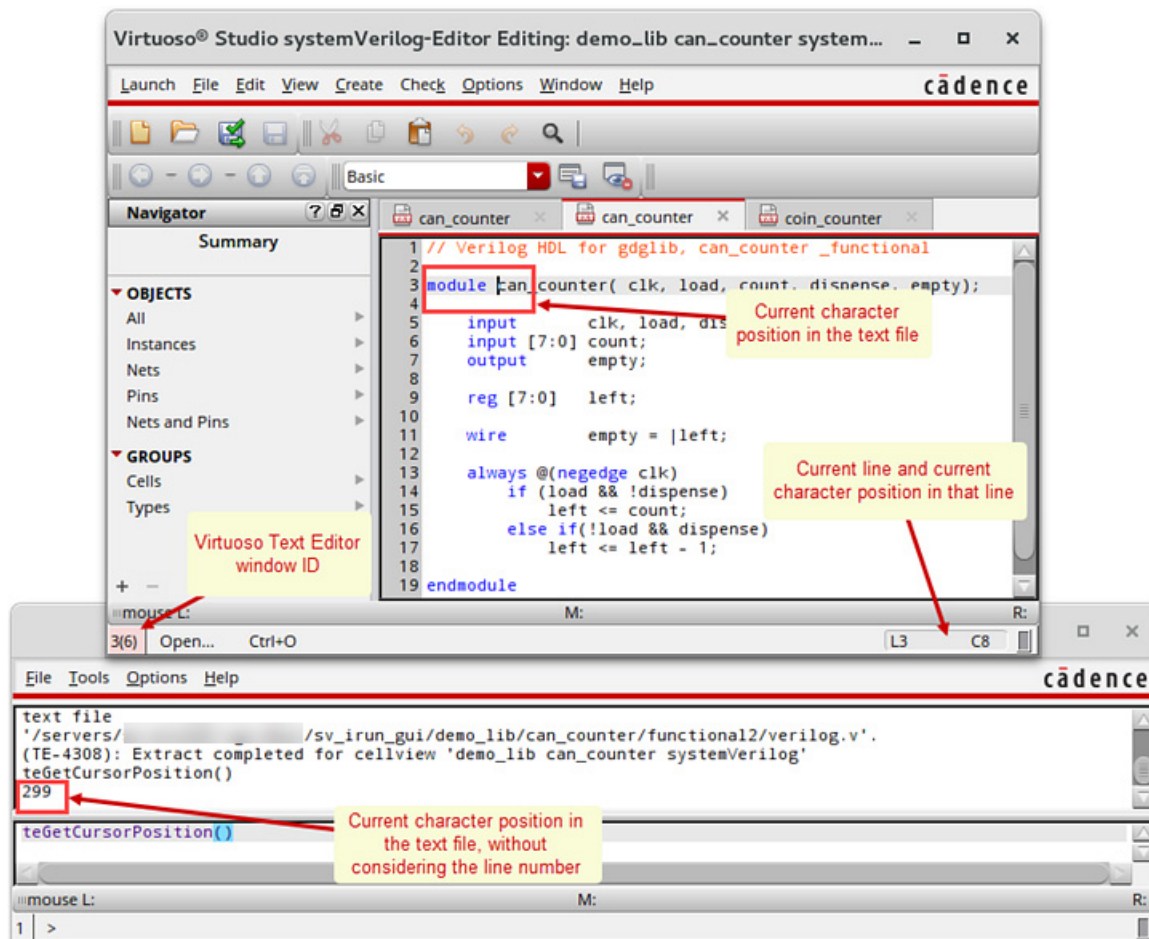
teGetCursorPosition

```
teGetCursorPosition(  
    [ g_windowId ]  
)  
=> charPosition / nil
```

Description

Returns the character position of the cursor in the current or specified Virtuoso Text Editor window.

The character position that this function returns is different from the line and character position displayed at the bottom-right side of the Virtuoso Text Editor window. The function counts all the characters before the current position starting from zero, and does not consider the line number. See the following figure.



Virtuoso Text Editor User Guide

Virtuoso Text Editor and HDL Package Setup Functions

Arguments

g_windowId The optional window ID of the Virtuoso Text Editor session.

Value Returned

charPosition The numeric character position of the cursor.

nil The character position is not retrieved because of an error.

Example

Consider that a text cellview is opened in Virtuoso Text Editor whose window ID is 2. The following are the first two lines of the text cellview:

```
//systemVerilog HDL for "demo_lib", "can_counter" "systemVerilog"
module can_counter ( clk, load, count, dispense, empty );
```

Place the cursor after `module` and just before `can_counter` on the second line. Notice that the line and character position displayed at the bottom-right corner of the window is L2 C8 for line 2 and character 8.

As illustrated below, run `teGetCursorPosition` from Virtuoso CIW to get the character position in the text cellview. Then run the same function to display the character position on window ID 2. Run the function using another window ID and notice the return value.

```
teGetCursorPosition()
73
teGetCursorPosition(window(2))
73
teGetCursorPosition(window(1))
nil
```

teIsModified

```
teIsModified(  
    d_cvId  
)  
=> t / nil
```

Description

Checks if a given text cellview been modified since last save.

Arguments

<i>d_cvId</i>	Cellview ID of the schematic which is being checked.
---------------	------------------------------------------------------

Value Returned

<i>t</i>	The text cellview has been modified by any active text editor.
<i>nil</i>	The cellview is not a text cellview or was not modified.

Example

Consider that the `myCellView` is the ID of the text cellview opened in Virtuoso Text Editor. You can check for the edits as shown here:

```
myCellView = ddGetObj("myLib" "myCell" "myVerilogView")  
when(teIsModified(myCellView) info("CellView is modified\n"))
```

teSave

```
teSave(  
    d_cvId  
)  
=> t / nil
```

Description

Saves the edits done in the text editor cellview.

Arguments

<i>d_cvId</i>	Cellview ID of the schematic which is being saved.
---------------	----------------------------------------------------

Value Returned

t	The text cellview has been saved.
nil	The text cellview has not been saved.

Example

Consider that the `myCellView` is the ID of the text cellview opened in Virtuoso Text Editor. You can save the edits as shown here:

```
myCellView = ddGetObj("myLib" "myCell" "myVerilogView")  
teSave(myCellView)
```

teSaveWithDerivedData

```
teSaveWithDerivedData(  
    d_cvId  
)  
=> t / nil
```

Description

Saves the in-memory content of the text editor open for the given cellview and generates the derived data (typically, creates a shadow database).

Arguments

<i>d_cvId</i>	Cellview ID of the schematic which is being saved.
---------------	----------------------------------------------------

Value Returned

<i>t</i>	The text cellview has been saved and derived data created successfully.
<i>nil</i>	The the cellView is not a text cellview, has not been saved, or the derived data could not be created.

Example

Consider that the `myCellView` is the ID of the text cellview opened in Virtuoso Text Editor. You can save the edits as shown here:

```
myCellView = ddGetObj("myLib" "myCell" "myVerilogView")  
teSaveWithDerivedData(myCellView)
```


teSetCursor

```
teSetCursor(  
    x_charPosition  
    [ g_windowId ]  
)  
=> t / nil
```

Description

Places the cursor on the specified character position in the current or specified Virtuoso Text Editor window.

Arguments

<i>x_charPosition</i>	The character position where you want to place the cursor.
<i>g_windowId</i>	The optional window ID of the Virtuoso Text Editor session.

Value Returned

<i>t</i>	The cursor is placed at the specified character position.
<i>nil</i>	The cursor is not placed because of an error.

Example

Consider that a text cellview is opened in Virtuoso Text Editor whose window ID is 2. The following are the first two lines of the text cellview:

```
//systemVerilog HDL for "demo_lib", "can_counter" "systemVerilog"  
module can_counter ( clk, load, count, dispense, empty );
```

Place the cursor after `module` and just before `can_counter` on the second line. As illustrated below, run `teGetCursorPosition` from Virtuoso CIW to get the character position. Run `teSetCursor` to place the cursor on the fifth character. Notice that the line and character position displayed at the bottom-right corner of the Virtuoso Text Editor window becomes L1 C6 for line 1 and character 6. Run `teGetCursorPosition` again. Now, place the cursor in the beginning of the text cellview using `teSetCursor` for the window ID 2.

```
teGetCursorPosition()  
73  
teSetCursor(5)  
t  
teGetCursorPosition()  
5  
teSetCursor(0 window(2))  
t
```

teSetEditMode

```
teSetEditMode(  
    d_cvId  
    d_mode  
)  
=> t / nil
```

Description

Sets the editing mode for the specified cellview to read-only(_r) or editable(_a) based on the value of the mode argument.

Arguments

<i>d_cvId</i>	Cellview ID of the schematic which is being set to the edit mode.
<i>d_mode</i>	The editing mode.

Value Returned

<i>t</i>	The edit mode of the text cellview changed successfully.
<i>nil</i>	The edit mode of the text cellview could not be changed.

Example

Consider that the `myCellView` is the ID of the text cellview opened in Virtuoso Text Editor. You can set the edit mode as shown here:

```
myCellView = ddGetObj("myLib" "myCell" "myVerilogView")  
teSetEditMode(myCellView "a")
```

teRegPostExtractTrigger

```
teRegPostExtractTrigger(  
    s_symbol  
)  
=> t / nil
```

Description

Registers a user-defined function that is triggered after the text editor extraction completes.

Arguments

- | | |
|-----------------|----------------------------------------|
| <i>s_symbol</i> | SKILL symbol for the trigger function. |
|-----------------|----------------------------------------|
- This trigger function further accepts the following two arguments:
- **ddId**: The ddId of the lib/cell/view being checked, which is fetched from the ddGetObj SKILL function.
 - **success**: A boolean, indicating success or failure.

Value Returned

- | | |
|-----|------------------------------------------|
| t | The trigger was registered successfully. |
| nil | The trigger could not be registered. |

Examples

The following example shows how the `ExampTextPostExtract` trigger procedure uses the `ddId` argument to retrieve the lib/cell/view names and the `success` argument to report a successful operation. The `viewType` and lib/cell/view names let you make the trigger function dependent upon the language used. The `teRegPostExtractTrigger` function is then used to register the trigger function.

```
procedure(ExampTextPostExtract(ddId success)  
    letseq(  
        ( (viewName ddId~>name)  
          (cellName ddId~>cell~>name)  
          (libName ddId~>lib~>name)  
          (masterId ddGetObj(libName cellName viewName "**"))  
          (viewType ddMapGetFileViewType(masterId))  
        )  
        printf("Library %L Cell %L View %L ViewType %L was extracted with success  
%L\n" libName cellName viewName viewType success )  
    )  
)  
teRegPostExtractTrigger('ExampTextPostExtract)
```

Related Topics

[teRegPreExtractTrigger](#)

teRegPreExtractTrigger

```
teRegPreExtractTrigger(  
    s_symbol  
)  
=> t / nil
```

Description

Registers a user-defined function that is triggered before the text editor extraction starts.

Arguments

<i>s_symbol</i>	SKILL symbol for the trigger function.
-----------------	----------------------------------------

This trigger function further accepts the following argument:

- **ddId**: The ddId of the lib/cell/view being checked, which is fetched from the ddGetObj SKILL function.

Value Returned

t	The trigger function was registered successfully.
nil	The trigger function could not be registered.

Examples

The following example shows how the `ExampTextPreExtract` trigger function uses the `ddID` argument to retrieve the `lib/cell/view` names. The `viewType` and `lib/cell/view` names let you make the trigger function dependent upon the language used. The `teRegPreExtractTrigger` function is then used to register the trigger function.

```
procedure ((ExampTextPreExtract (ddId)  
    letseq(  
        ( (viewName ddId~>name)  
        (cellName ddId~>cell~>name)  
        (libName ddId~>lib~>name)  
        (masterId ddGetObj(libName cellName viewName "*"))  
        (viewType ddMapGetFileViewType(masterId))  
    )  
    printf("Library %L Cell %L View %L ViewType %L will be extracted\n" libName  
        cellName viewName viewType )  
    )  
)  
teRegPostExtractTrigger('ExampTextPreExtract)
```

Related Topics

[teRegPostExtractTrigger](#)

teUnRegPostExtractTrigger

```
teUnRegPostExtractTrigger(  
    s_symbol  
)  
=> t / nil
```

Description

Unregisters functions previously registered by you so that they are no longer triggered after the text editor functionality is being run.

Arguments

<i>s_symbol</i>	SKILL symbol for the trigger function.
-----------------	----------------------------------------

Value Returned

t	The trigger was unregistered successfully.
nil	The trigger could not unregister.

Example

You can unregister post trigger as shown here:

```
;; define post trigger  
defun( postTrigger (cvId success)  
    info("*** postTrigger called with lcv = '%s %s %s' and success = %L\n"  
        cvId~>parent~>parent~>name cvId~>parent~>name cvId~>name success)  
    ;; The success can have one of the following values: t, nil, or unknown.  
    ;; 'unknown' refers to the scenario when there are warnings displayed but cannot  
    determine if the operation was successful or not.  
    t  
    )  
  
;; register post trigger  
teRegPostExtractTrigger('postTrigger)  
  
;; unregister post trigger  
teUnRegPostExtractTrigger('postTrigger)
```

teUnRegPreExtractTrigger

```
teUnRegPreExtractTrigger(  
    s_symbol  
)  
=> t / nil
```

Description

Unregisters functions previously registered by you so that they are no longer triggered prior to running the text editor functionality.

Arguments

<i>s_symbol</i>	SKILL symbol for the trigger function.
-----------------	----------------------------------------

Value Returned

t	The trigger was unregistered successfully.
nil	The trigger could not unregister.

Example

You can unregister pre trigger as shown here:

```
;; define pre trigger  
defun( preTrigger (cvId)  
    info("*** preTrigger called with lcv = '%s %s %s'\n"  
        cvId~>parent~>parent~>name cvId~>parent~>name cvId~>name)  
    t  
)  
  
;; register pre trigger  
teRegPreExtractTrigger('preTrigger)  
  
;; unregister pre trigger  
teUnRegPreExtractTrigger('preTrigger)
```


hdlGenerateTextDatabase

```
hdlGenerateTextDatabase(  
    t_libName  
    t_cellName  
    t_viewName  
)  
=> t / nil
```

Description

Generates the text database for analog and digital languages without requiring the Text Editor.

Arguments

<i>t_libName</i>	Specifies the library name of the cellview.
<i>t_cellName</i>	Specifies the name of the cellview.
<i>t_viewName</i>	Specifies the view name of the cellview.

Value Returned

<i>t</i>	If the database is generated successfully.
<i>nil</i>	If the database generation failed.

Example

To generate the text database for a given cell and print terminals:

```
when(hdlGenerateTextDatabase("myLib" "myCell" "myView")  
    cv = dbOpenCellViewByType("myLib" "myCell" "myView")  
    when(cv  
        cv~>terminals~>name  
    )  
)
```

Virtuoso Text Editor User Guide

Virtuoso Text Editor and HDL Package Setup Functions

Environment Variables

This appendix describes the environment variables of Virtuoso Text Editor and HDL Package Setup form that you can use to customize the default settings. You can set the values of these environment variables in the `.cdsenv` or `.cdsinit` file. The following example illustrates how an environment variable is set in the `.cdsenv` file.

```
textedit      useExternalEditor      boolean      nil
```

For details on the `.cdsenv` file, see *[Specifying Environment Settings](#)* in *Virtuoso Design Environment User Guide*.

This appendix describes the following Virtuoso Text Editor and HDL Package Setup form environment variables:

- CDS5X_NOLINK
- enableAhdllintStaticCheck
- hdlPkgAdvancedMode
- hdlPkgEnableHdlSetup
- hdlPkgEnableUPF
- hdlPkgEnableUVM
- hdlPkgHdlvarFile
- hdlPkgMakeLibTree
- hdlPkgProjDir
- hdlPkgUseHdlvar
- hdlPkgUseXmlbdirname
- hdlPkgUVMVersion
- hdlPkgXmlbdirname

- hdlPkgXrunArgsFileTableList
- inScopeReadBGColor
- inScopeWriteBGColor
- showLineNumbersInSideBar
- syntaxLineLength
- useExternalEditor
- ignoreDesignValCheckXcelium
- messageSeverityXcelium
- designCheckLogFileXcelium
- messageSeverityExtractionFramework

Note: Only the environment variables documented in this section are supported for public use. Any other environment variables and undocumented aspects of the environment variables described below are private and subject to change or removal at any time.

CDS5X_NOLINK

If set to `t`, the netlist file is copied in the cellview. Else, a symbolic link of the netlist file is created in the cellview. This variable is useful in creating a symbolic link when the size of the netlist file is large.

Variable Type	Boolean
Default Value	<code>t</code>
Acceptable Values	<code>t, nil</code>

enableAhdllintStaticCheck

If set to `t`, enables AHDL Linter static check in Virtuoso Text Editor.

Variable Type	Boolean
Default Value	<code>nil</code>
Acceptable Values	<code>t, nil</code>

Related Topics

[Understanding the Graphical User Interface](#)

hdlPkgAdvancedMode

Displays the options used to specify additional package text files and name of the `xcelium.d` directory for `xrun` compilation in the Virtuoso HDL Package Setup form. By default, these options are not displayed.

Variable Type	toggle
Default Value	<code>nil</code>
Acceptable Values	<code>t, nil</code>

GUI Equivalent

Command	<i>Tools – AMS – HDL Package Setup</i>
Field	<i>Advanced mode</i>

Examples

```
envSetVal("hdlPkg" "hdlPkgAdvancedMode" 'toggle' (nil))
```

Related Topics

[Virtuoso HDL Package Setup Form](#)

hdlPkgEnableHdlSetup

Enables the flow where `xrun` is called with the options specified on the Virtuoso HDL Package Setup form to compile HDL files. A `*.pak` file is created in the default scratch directory instead of the existing Virtuoso library directory. By default, the flow is not enabled.

Variable Type	Toggle
Default Value	<code>nil</code>
Acceptable Values	<code>t, nil</code>

GUI Equivalent

Command	<i>Tools – AMS – HDL Package Setup</i>
Field	<i>Enable HDL package setup</i>

Examples

```
envSetVal("hdlPkg" "hdlPkgEnableHdlSetup" 'toggle' (nil))
```

Related Topics

[Virtuoso HDL Package Setup Form](#)

hdlPkgEnableUPF

Enables the SystemVerilog UPF package. By default, the SystemVerilog UPF package is not enabled.

Variable Type	toggle
Default Value	nil
Acceptable Values	t, nil

GUI Equivalent

Command	<i>Tools – AMS – HDL Package Setup</i>
Field	<i>Enable SystemVerilog UPF</i>

Examples

```
envSetVal("hdlPkg" "hdlPkgEnableUPF" 'toggle' (nil))
```

Related Topics

[Virtuoso HDL Package Setup Form](#)

hdlPkgEnableUVM

Enables the UVM compilation built-in package. By default, the UVM compilation package is not enabled.

Variable Type	toggle
Default Value	nil
Acceptable Values	t, nil

GUI Equivalent

Command	<i>Tools – AMS – HDL Package Setup</i>
Field	<i>Enable UVM Compilation</i>

Examples

```
envSetVal("hdlPkg" "hdlPkgEnableUVM" 'toggle ' (nil))
```

Related Topics

- [Virtuoso HDL Package Setup Form](#)
- [hdlPkgUVMVersion](#)

hdlPkgHdlvarFile

Specifies the name and absolute path of an `hdl.var` file. The is recommended only when the project uses `hdl.var` legacy file. By default, no `hdl.var` file is specified.

Variable Type	string
Default Value	" "
Acceptable Values	string specifying the name and path of a valid hdl.var file enclosed in double quotes

GUI Equivalent

Command	<i>Tools – AMS – HDL Package Setup</i>
Field	<i>hdl.var file</i>

Examples

```
envSetVal("hdlPkg" "hdlPkgHdlvarFile" 'string "$Usr/hdl.var")
```

Related Topics

- [Virtuoso HDL Package Setup Form](#)
- [hdlPkgUseHdlvar](#)

hdlPkgMakeLibTree

Creates the [makelib table](#).

Variable Type	String
Default Value	" "
Acceptable Values	string specifying the values of all the columns in the table enclosed in double quotes

GUI Equivalent

Command	<i>Tools – AMS – HDL Package Setup</i>
Field	<i>makelib (table)</i>

Examples

```
envSetVal("hdlPkg" "hdlPkgMakeLibTree" 'string "((t \"pkgLib\" \"$PROJ/  
package_1.sv\" \"-incdir$PROJ/pkg_source\") (t \"pkgLib\" \"$PROJ/  
package_2.sv\" \"\"\"\"))")
```

Related Topics

[Virtuoso HDL Package Setup Form](#)

hdlPkgProjDir

Specifies the directory to save the `xcelium.d` and `xcelium` compilation results.

Variable Type	string
Default Value	<code>"./compileScratch"</code>
Acceptable Values	string specifying the path to any valid, accessible file system location

GUI Equivalent

Command	<i>Tools – AMS – HDL Package Setup</i>
Field	<i>Default scratch directory</i> (text field)

Examples

```
envSetVal("hdlPkg" "hdlPkgProjDir" 'string "./compileScratch_test")
```

Related Topics

[Virtuoso HDL Package Setup Form](#)

hdlPkgUseHdlvar

Enables the option to specify an `hdl.var` file. By default, the option is disabled.

Variable Type	toggle
Default Value	<code>nil</code>
Acceptable Values	<code>t, nil</code>

GUI Equivalent

Command	<i>Tools – AMS – HDL Package Setup</i>
Field	<i>hdl.var file t</i>

Examples

```
envSetVal("hdlPkg" "hdlPkgUseHdlvar" 'toggle' (nil))
```

Related Topics

- [Virtuoso HDL Package Setup Form](#)
- [hdlPkgHdlvarFile](#)

hdlPkgUseXmlbdirname

Enables the option to specify name of the `xcelium.d` directory. By default, the directory is not specified.

Variable Type	<code>toggle</code>
Default Value	<code>nil</code>
Acceptable Values	<code>t, nil</code>

GUI Equivalent

Command	<i>Tools – AMS – HDL Package Setup</i>
Field	<i>-xmlbdirname</i>

Examples

```
envSetVal("hdlPkg" "hdlPkgUseXmlbdirname" 'toggle' (nil))
```

Related Topics

[Virtuoso HDL Package Setup Form](#)

hdlPkgUVMVersion

Specifies the UVM version if UVM compilation is enabled.

Variable Type	string
Default Value	"1.2"
Acceptable Values	String specifying any valid UVM version

GUI Equivalent

Command	<i>Tools – AMS – HDL Package Setup</i>
Field	<i>UVM version</i>

Examples

```
envSetVal("hdlPkg" "hdlPkgUVMVersion" 'string "1.2")
```

Related Topics

- [Virtuoso HDL Package Setup Form](#)
- [hdlPkgEnableUVM](#)

hdlPkgXmlbdirname

Specifies the name of the `xcelium.d` directory.

The specified name is used together with the value of the `-xmlbdirpath` argument, therefore, must not contain any path information.

Variable Type	string
Default Value	" "
Acceptable Values	String specifying a valid directory on the file system

GUI Equivalent

Command	<i>Tools – AMS – HDL Package Setup</i>
Field	<i>-xmlibdirname</i>

Examples

```
envSetVal("hdlPkg" "hdlPkgXmlibdirname" 'string "myXmlLibDir")
```

Related Topics

[Virtuoso HDL Package Setup Form](#)

hdlPkgXrunArgsFileTableList

Creates the [Options files table](#).

Variable Type	String
Default Value	" "
Acceptable Values	string specifying the values of all the columns in the table enclosed in double quotes

GUI Equivalent

Command	<i>Tools – AMS – HDL Package Setup</i>
Field	<i>Options files (table)</i>

Examples

```
envSetVal("hdlPkg" "hdlPkgXrunArgsFileTableList" 'string "((t \" $PROJ/  
test_compile.f\") (t \" $PROJ/pkgs.f\"))")
```

Related Topics

[Virtuoso HDL Package Setup Form](#)

inScopeReadBGColor

Sets the background color of the content when a text cellview is opened in read-only mode.

Variable Type	String
Default Value	"#E8E8E8"
Acceptable Values	String, nil

Related Topics

[Switching Between Edit Mode and Read-Only Mode](#)

inScopeWriteBGColor

Sets the background color of the content when a text cellview is opened in edit mode.

Variable Type	String
Default Value	"white"
Acceptable Values	String, nil

Related Topics

[Switching Between Edit Mode and Read-Only Mode](#)

showLineNumbersInSideBar

If set to `t`, Virtuoso Text Editor displays the line number of each line in the text cellview file.

Variable Type	Boolean
Default Value	<code>t</code>
Acceptable Values	<code>t</code> , <code>nil</code>

Related Topics

[Showing Line Numbers and Going to a Line Number](#)

syntaxLineLength

Specifies the number of characters that will be processed by the syntax highlighter on every line.

Variable Type	Integer
Default Value	1000
Acceptable Values	0-1000

useExternalEditor

If set to `t`, Virtuoso uses an external editor as the default editor instead of using Virtuoso Text Editor.

Variable Type	Boolean
Default Value	<code>nil</code>
Acceptable Values	<code>t</code> , <code>nil</code>

Related Topics

[Editing a Text Cellview File in an External Editor](#)

ignoreDesignValCheckXcelium

```
textif ignoreDesignValCheckXcelium boolean nil nil
```

Description

Ignores the Xcelium design validation checks. The default value is `nil`, which indicates that the Xcelium design validation checks are done.

GUI Equivalent

None

Examples

```
envGetVal("textif" "ignoreDesignValCheckXcelium")  
envSetVal("textif" "ignoreDesignValCheckXcelium" 'boolean t)
```

messageSeverityXcelium

```
textif messageSeverityXcelium cyclic {"AllCurrentVersion" | "Notice" | "Warning" |  
    "Error" | "System" | "Internal" | "None" | "AllAnyVersion"}
```

Description

Controls the severity of the Xcelium design validation checks. The default value is `None`, which indicates that no errors or warnings for Xcelium design validation checks are reported. Possible values are:

- `AllCurrentVersion`: All messages for the current version of Xcelium are reported.
- `Notice`: Only messages of type `Notice` are reported.
- `Warning`: Only messages of type `Warning` are reported.
- `Error`: Only messages of type `Error` are reported.
- `System`: Only messages of type `System` are reported.
- `Internal`: Only messages of type `Internal` are reported.
- `None`: No messages are reported.
- `AllAnyVersion`: All messages for all versions are reported.

GUI Equivalent

None

Examples

```
envGetVal("textif" "messageSeverityXcelium")  
envSetVal("textif" "messageSeverityXcelium" 'cyclic "AllCurrentVersion")  
envSetVal("textif" "messageSeverityXcelium" 'cyclic "Notice")  
envSetVal("textif" "messageSeverityXcelium" 'cyclic "Warning")  
envSetVal("textif" "messageSeverityXcelium" 'cyclic "Error")  
envSetVal("textif" "messageSeverityXcelium" 'cyclic "System")  
envSetVal("textif" "messageSeverityXcelium" 'cyclic "Internal")  
envSetVal("textif" "messageSeverityXcelium" 'cyclic "AllAnyVersion")
```


designCheckLogFileXcelium

```
textif designCheckLogFileXcelium string "./vamsXceliumDesignValidationLogs"
```

Description

Specifies the path of the log file in which the messages are printed.

The default value is `"./vamsXceliumDesignValidationLogs"`.

GUI Equivalent

None

Examples

```
textif designCheckLogFileXcelium string "/user1/myValidationLogs"
```

messageSeverityExtractionFramework

```
textif messageSeverityExtractionFramework cyclic { "All" | "Warning" | "Error" |  
    "System" | "Failure" | "AllExceptWarning" | "None" }
```

Description

Controls the severity of the messages reported by the Xcelium extraction framework. The default value is `All`, which indicates that all errors or warnings for Xcelium design validation checks are reported. Possible values are:

- `All`: Default. All messages are reported.
- `Warning`: Only messages of type `Warning` are reported.
- `Error`: Only messages of type `Error` are reported.
- `System`: Only messages of type `System` are reported.
- `Failure`: Only messages of type `Failure` are reported.
- `AllExceptWarning`: All messages except those of type `Warning` are reported.
- `None`: No messages are reported.

GUI Equivalent

None

Examples

```
envGetVal("textif" "messageSeverityExtractionFramework")  
envSetVal("textif" "messageSeverityExtractionFramework" 'cyclic "Warning")  
envSetVal("textif" "messageSeverityExtractionFramework" 'cyclic "Error")  
envSetVal("textif" "messageSeverityExtractionFramework" 'cyclic "System")  
envSetVal("textif" "messageSeverityExtractionFramework" 'cyclic "Failure")  
envSetVal("textif" "messageSeverityExtractionFramework" 'cyclic  
"AllExceptWarning")  
envSetVal("textif" "messageSeverityExtractionFramework" 'cyclic "None")
```

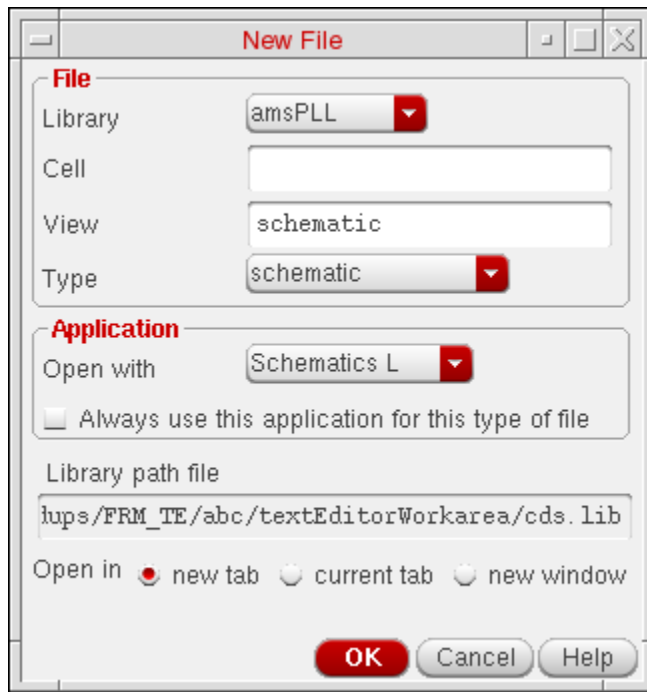
Virtuoso Text Editor Forms

This appendix describes the following Virtuoso Text Editor forms:

- [New File Form](#) on page 84
- [Open File Form](#) on page 86
- [Cellview From Cellview Form](#) on page 87
- [Find and Replace Form](#) on page 89
- [Go to Line Form](#) on page 90
- [Text Editor Options Form](#) on page 90
- [Add Bookmark and Bookmark Manager Forms](#) on page 91
- [Workspaces Forms](#) on page 91

New File Form

The New File form lets you create a new cellview. You can also invoke this form from other Virtuoso applications, such as Virtuoso Library Manager and Virtuoso Schematic Editor.



Form Component	Description
<i>Library</i>	Choose the name of the library where you want to create a new cellview.
<i>Cell</i>	Type a cell name for the new cellview.
<i>View</i>	Specify the view name for the new cellview. This field displays the default view name associated with the cell type, which you can edit.
<i>Type</i>	Choose the type of view to be created. Based on the selection, the <i>View</i> and <i>Open with</i> fields get updated.
<i>Open with</i>	Choose the application that is invoked to display the new cellview.

Virtuoso Text Editor User Guide

Virtuoso Text Editor Forms

Form Component	Description
<i>Always use this application for this type of file</i>	Select to make the chosen application as the default application to open the specified type of cellview.
<i>Open in</i>	Select the location where you want to open the new cellview. You can open the cellview in a new or same tab in the window, or in a new window.

Related Topics

[Creating a Text Cellview](#)

[Opening a New Cellview](#) in *Virtuoso Schematic Editor L User Guide*

[Creating a New Cellview](#) in *Cadence Library Manager User Guide*

Open File Form

The Open File form lets you open a cellview. You can also invoke this form from other Virtuoso applications, such as Virtuoso Library Manager and Virtuoso Schematic Editor.

Form Component	Description
<i>Library</i>	Choose the name of the library that contains the cellview you want to open. The <i>Cells</i> area updates to display the cells available in the library.
<i>Cell</i>	Type the cell name or select the name from the <i>Cells</i> area.
<i>View</i>	Choose one of the available views of the cell you want to open. The <i>Type</i> field updates to display the type of the selected view.
<i>Browse</i>	Click to open the Library Browser - Open File form to select the library, cell, and view, instead of specifying them in their respective fields. The fields update to reflect the selection.
<i>Open with</i>	Choose the application that is invoked to display the new cellview.

Virtuoso Text Editor User Guide

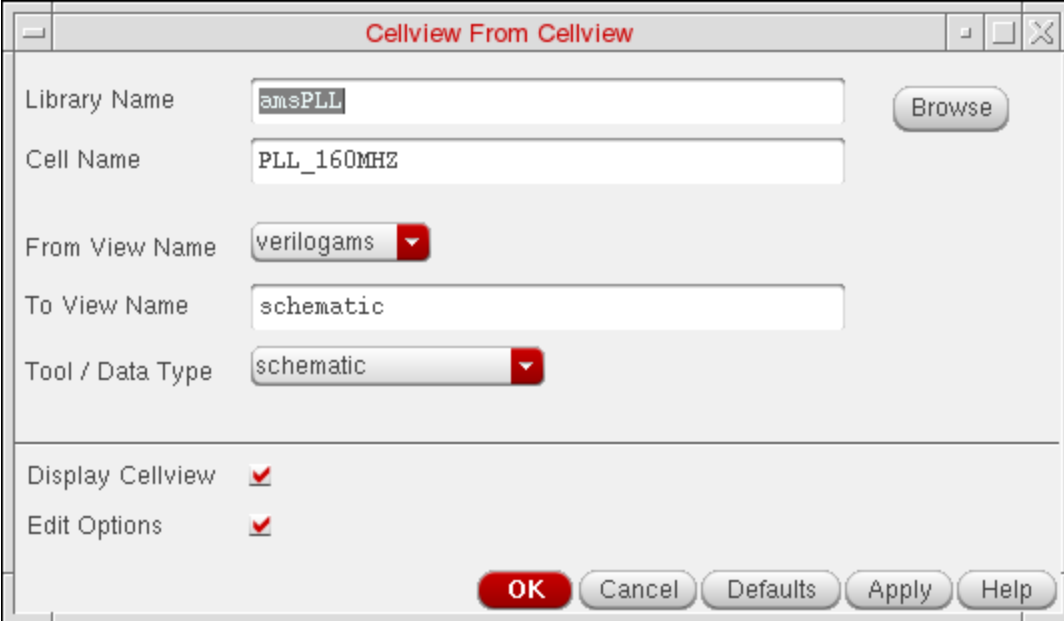
Virtuoso Text Editor Forms

Form Component	Description
<i>Always use this application for this type of file</i>	Select to make the chosen application as the default application to open the specified type of cellview.
<i>Open for</i>	Select <i>edit</i> to open the cellview in edit mode, or <i>read</i> to open it in read-only mode.
<i>Open in</i>	Select the location where you want to open the cellview. You can open the cellview in a new or same tab in the window, or in a new window.

Reference: [“Launching Virtuoso Text Editor”](#) on page 9

Cellview From Cellview Form

The Cellview From Cellview form lets you create a cellview using another cellview as a source.



The screenshot shows the 'Cellview From Cellview' dialog box. It has a title bar with the text 'Cellview From Cellview'. Inside the dialog, there are several input fields and buttons. The 'Library Name' field contains 'amsPLL' and has a 'Browse' button next to it. The 'Cell Name' field contains 'PLL_160MHZ'. The 'From View Name' field is a dropdown menu showing 'verilogams'. The 'To View Name' field contains 'schematic'. The 'Tool / Data Type' field is a dropdown menu showing 'schematic'. At the bottom, there are two checked checkboxes: 'Display Cellview' and 'Edit Options'. At the very bottom, there are five buttons: 'OK' (highlighted in red), 'Cancel', 'Defaults', 'Apply', and 'Help'.

Virtuoso Text Editor User Guide

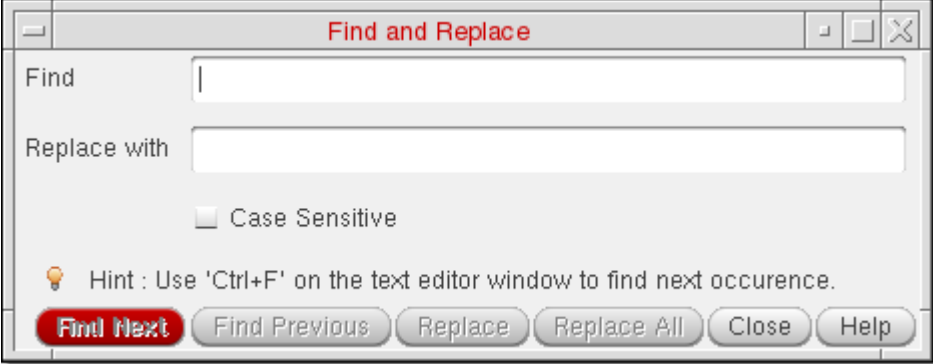
Virtuoso Text Editor Forms

Form Component	Description
<i>Library Name</i>	Specify the library name of the source cellview. Note: The library, cell, and view of the currently opened text cellview appear in the respective fields, which you can change. You can also click <i>Browse</i> and choose the library, cell, and view from Library Browser.
<i>Cell Name</i>	Specify the cell name of the source cellview.
<i>From the View Name</i>	Select the source view name.
<i>To View Name</i>	Specify the destination view name.
<i>Tool / Data Type</i>	Select the destination view type. The <i>To View Name</i> field updates with the default view name of the selected type.
<i>Display Cellview</i>	Select to open the new cellview in a new window.
<i>Edit Options</i>	Select to edit any additional options before the cellview creation.

Reference: [“Creating a Cellview from an Existing Cellview”](#) on page 23

Find and Replace Form

The Find and Replace form lets you locate and replace text in the cellview opened in Virtuoso Text Editor.

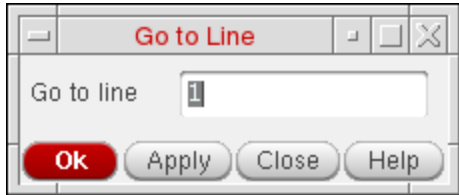


Form Component	Description
<i>Find</i>	Type the text that you want to search in the file currently opened in Virtuoso Text Editor. All occurrences of the text get highlighted.
<i>Replace with</i>	Type the replacement text, if required. To replace text, use the <i>Replace</i> or <i>Replace All</i> buttons.
<i>Case Sensitive</i>	Select if you want to search for the text that matches the case of the text you typed in the <i>Find</i> field.
Find and Replace buttons	Use the appropriate button to find the text specified in the <i>Find</i> field, or replace the text specified in the <i>Find</i> field with the text specified in the <i>Replace with</i> field. The available buttons are <i>Find Next</i> , <i>Find Previous</i> , <i>Replace</i> , and <i>Replace All</i> . Press <code>Ctrl+F</code> to go to the next occurrence of the text.

Reference: [“Using Other Editing Features”](#) on page 33

Go to Line Form

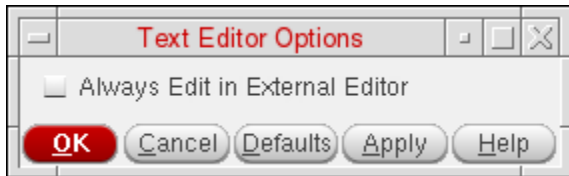
The Go to Line form lets you go to a specific line number in the displayed file.



Reference: [“Showing Line Numbers and Going to a Line Number”](#) on page 28

Text Editor Options Form

The Text Editor Options form lets you set certain default behavior of Virtuoso Text Editor.



Form Component	Description
<i>Always Edit in External Editor</i>	Select to open text cellviews for editing in an external editor. To revert this setting, open the text cellview in read-only mode in Virtuoso Text Editor and reset the option.

References:

- [“Editing a Text Cellview File in an External Editor”](#) on page 30
- [“useExternalEditor”](#) on page 78

Add Bookmark and Bookmark Manager Forms

The Add Bookmark and Bookmark Manager forms let you manage bookmarks for quick access to the book-marked items.

Reference: [Add Bookmark Form](#)

Workspaces Forms

The following workspaces forms let you manage the Virtuoso Text Editor workspaces:

- Save Workspace form
- Delete Workspace form
- Load Workspace form
- Set Default Workspace form

You can access these forms from the *Window — Workspaces* menu options.

Reference: [Virtuoso Workspaces](#)

Virtuoso Text Editor User Guide
Virtuoso Text Editor Forms

HDL Package Setup

This topic describes the Virtuoso HDL Package Setup form. The package files are important in SystemVerilog, SystemVerilog-AMS, VHDL, and VHDL-AMS for sharing and reusing common code definitions, such as functions or custom data types, between models. The form ensures that handling HDL package files becomes more intuitive, requires less maintenance, and improves the task flow for SystemVerilog, SystemVerilog-AMS, VHDL, and VHDL-AMS package files.

The Virtuoso HDL Package Setup form has the following benefits.

- Applies the unified HDL package text file setup for all Virtuoso tools.
- Defines the compilation order for the HDL package text files.
- Searches the HDL package text views in all `cds.lib` libraries.
- Checks HDL package setup and displays dependencies.
- Exports HDL package setup as `xrunArgs` file for reuse.

Related Topics

[Accessing the Virtuoso HDL Package Setup Form](#)

[Virtuoso HDL Package Setup Form](#)

[Use of HDL Package Settings in Various Tools](#)

Virtuoso Text Editor User Guide

HDL Package Setup

Accessing the Virtuoso HDL Package Setup Form

To access the Virtuoso HDL Package Setup form:

1. Click *Tools – AMS – HDL Package Setup*.

The Virtuoso HDL Package Setup form appears.

The screenshot shows the 'Virtuoso HDL Package Setup' dialog box. It has a title bar with the text 'Virtuoso HDL Package Setup'. The main area is divided into several sections:

- Enable HDL package setup:** A checkbox that is currently unchecked.
- Built-In Package Setup:** Contains two checkboxes: 'Enable UVM compilation' (unchecked) and 'Enable SystemVerilog UPF' (unchecked). To the right of these is a 'UVM version' dropdown menu set to '1.2'.
- Options:** Contains three buttons: 'Save Settings', 'Load Settings', and 'Export xrunArgs'.
- On/Off | Include Options:** A table with two columns. The first column is 'On/Off' and the second is 'Include Options'. There is a single row with a folder icon and the text '<Click here to add options files>'. To the right of the table are three buttons: a plus icon, a minus icon, and a close icon (X).
- Scratch directory:** A text field containing './compileScratch' and a button with three dots to its right. To the right of the text field is a button labeled 'Open Terminal'.
- Use hdl.var file:** A checkbox that is unchecked, followed by a text field and a button with three dots to its right.
- Buttons:** Below the 'Use hdl.var file' section are two buttons: 'Display hdl.var File Used by xrun' and 'Clear Compilation Results'.
- Advanced mode:** A checkbox that is checked, labeled 'Advanced mode'.
- Local Package Setup:** A section with a button 'Load HDL Package Cellviews From cds.lib Libraries'. Below it is a 'Package loading mode' section with two radio buttons: 'Append' (selected) and 'Overwrite'. Below this is a table with four columns: 'On/Off', 'Library Name', 'Source File', and 'Options'. There is a single row with a folder icon, the text '-makelib', and '<Click here to add package files>'. To the right of the table are three buttons: a plus icon, a minus icon, and a close icon (X).
- Virtuoso cds.lib file:** A text field containing 'servers/scratch05/rgirdhar/multiBlockMultiDSPF_old/cds.lib' and a button labeled 'Check cds.lib'.
- xmlbdirname:** A checkbox that is unchecked, followed by a text field.
- Buttons:** Below the '-xmlbdirname' section are three buttons: 'Compile Package Files', 'View Log', and 'Open Package Checking Viewer'.
- Footer:** At the bottom of the dialog are four buttons: 'OK' (highlighted in red), 'Cancel', 'Apply', and 'Help'.

2. Select the *Enable HDL package setup* check box to implement the HDL package settings defined in this form.
3. Enable the required options for the following tools:
 - ☐ ADE
 - ☐ *Enable the HDL Package Setup.*
 - ☐ *Enable the Reuse HDL package setup* option on the *Main* tab of the AMS Option form.
 - ☐ Virtuoso SystemVerilog Netlister
 - ☐ *Enable the HDL Package Setup.*
 - ☐ *Enable the Reuse HDL package setup* option in the SystemVerilog Netlister Options form.

Related Topics

[HDL Package Setup](#)

[Virtuoso HDL Package Setup Form](#)

[Use of HDL Package Settings in Various Tools](#)

Use of HDL Package Settings in Various Tools

The Virtuoso HDL Package Setup form can be used to define the settings for the following tools.

Tools	Uses
Text Editor	Always used.
Hierarchy Editor	Automatically used through text view check and saved when creating the <code>pc.db</code> information.
ADE/UNL	When enabled through the <i>Reuse HDL package setup</i> option on the <i>Main</i> tab of the AMS Option form, the package setup information is added to the regular netlist and xrun simulation options.
ADE/Simulation	Through <code>maestro</code> views. The status of the <i>Reuse HDL package setup</i> option on the <i>Main</i> tab of the AMS Option form is stored in the <code>maestro</code> view.
runams	Through <code>maestro</code> views. The status of the <i>Reuse HDL package setup</i> option on the <i>Main</i> tab of the AMS Option form is stored in the <code>maestro</code> view.
Virtuoso SystemVerilog Netlister	Always used if you enable the <i>Reuse HDL package setup</i> option in the SystemVerilog Netlister Options form.

Virtuoso Text Editor User Guide

HDL Package Setup

Tools

cdsTextTo5x

Uses

Only the xrunArgs file exported from the Virtuoso HDL Package Setup form is added to `cdsHDL_XrunArgsPKG.f`.

Note: By default, cdsTextTo5x does not implement the Virtuoso HDL Package Setup flow. However, you can implement the Virtuoso HDL Package Setup flow as shown below:

1. Export the `xrunArgsPKGexport.f` from the Virtuoso HDL Package Setup form.
2. From the Linux command line, edit the `xrunArgsPKGexport.f` by commenting out `-cdslib`.
3. Save the file.
4. Run the following commands at the Linux command line:

- a. `setenv ProjDir `pwd``
- b. `setenv XRUNOPTS "-f $PWD/xrunArgsPKGexport.f"`
- c. `cdsTextTo5x -CDSLIB cds.lib -LANG systemverilog -LIB lib1 -CELL mydesign -VIEW systemVerilog mydesign.sv`

You see the following output:

```
Virtuoso Framework License (111) was checked out
successfully. Total checkout time was 0.04s.
INFO (CDS5X-3014): Successfully created 5x structure
for cellview '(lib1 mydesign systemVerilog)'
```

xrun command line

Only the xrunArgs file exported from the Virtuoso HDL Package Setup form is added to `cdsHDL_XrunArgsPKG.f`.

Related Topics

[HDL Package Setup](#)

Virtuoso Text Editor User Guide

HDL Package Setup

Virtuoso HDL Package Setup Form

Use of HDL Package Settings in Various Tools


Virtuoso HDL Package Setup Form

This topics describes the details of the form fields.

Field Name	Description
<i>Enable HDL package setup</i>	When selected, enables the flow where <code>xrun</code> is called with the options specified on the Virtuoso HDL Package Setup form to compile HDL files. A <code>*.pak</code> file is created in the default scratch directory instead of the existing Virtuoso library directory.
Built-In Package Setup	This section lets you specify the built-in package settings that are applicable for the HDL package setup flow.
<i>Enable UVM compilation</i>	When selected, adds <code>-uvm -uvmhome CDNS-1.2</code> as one of the <code>xrun</code> compilation options.
<i>UVM version</i>	Specifies the UVM version from the drop-down list.
<i>Enable SystemVerilog UPF</i>	When selected, adds <code>-makelib worklib -sv `xmroot`/tools/inca/files/1801/upf_package.sv</code> as one of the <code>xrun</code> compilation options.
Options	This section lets you specify various options that can be set for the HDL package setup flow.
<i>Save Settings</i>	Saves the Virtuoso HDL Package Setup form settings to a specified file.
<i>Load Settings</i>	Loads settings from the specified file. The file specified must be an unmodifiable file saved using <i>Save Settings</i> .
<i>Export XrunArgs</i>	Exports the <code>xrunArgs</code> file based on the Virtuoso HDL Package Setup form settings. Alternatively, you can use the <code>hdlPkgExportXrunArgs</code> function in Virtuoso CIW.
<i>Export Environment Variables</i>	Opens the Export Environment Variables form to let you export all HDL Package Setup form settings as environment variables.
Options files table	
<i>On/Off</i>	Enables or disables the use of the HDL package definition in this row.


Virtuoso Text Editor User Guide

HDL Package Setup

Field Name	Description
<i>Include Options</i>	<p>Specifies xrunArgs files, which contain user-defined xrun compilation options.</p> <p>Any package files specified in the <code>-f</code> file must be surrounded with <code>-makelib</code> and <code>-endlib</code> to trigger the pre-compilation.</p>  <p>After the xrunArgs file is specified, the <i>Edit</i> button is enabled. You can click it to open and edit the file.</p>
<i>Scratch directory</i>	<p>Specifies a directory to generate the <code>xcelium.d</code> and <code>xcelium</code> compilation results.</p> <p>It is a mandatory field, and the default value is <code>./compileScratch</code>. If the directory specified does not exist, the tool creates the directory automatically before compiling.</p>
<i>Open Terminal</i>	Launch xterm with the LINUX path specified in the <i>Scratch directory</i> field.
<i>Use hdl.var file</i>	Specifies an <code>hdl.var</code> file. Recommended only when there is a project <i>hdl.var</i> legacy.
<i>Display hdl.var File Used by xrun</i>	Displays the <code>hdl.var</code> file.
<i>Clear Compilation Results</i>	Removes the Xcelium compilation results by running command <code>xrun -clean -xmlibdirpath ./compileScratch</code> .
<i>Advanced mode</i>	When selected, specifies additional package text files and <code>-xmlibdirname</code> for xrun compilation.
Local Package Setup	This section lets you specify the local package settings that are used in the HDL package setup flow.
<i>Load HDL Package Cellviews From cds.lib Libraries</i>	<p>Searches all libraries defined in <code>cds.lib</code> for package cellviews that have the following view types:</p> <ul style="list-style-type: none"> ■ <code>systemVerilogPackageText</code> ■ <code>vhdl</code> and <code>VHDLAMSText</code> and view name as <code>package</code> or <code>body</code>
<i>Package loading mode</i>	Defines the package files loading mode after the search is completed.
<i>Append</i>	Merges results after comparing with the existing makelib list and appends only the additional ones to the list.
<i>Overwrite</i>	Overwrites the existing makelib list. This option is useful when starting with a new setup or resetting entries.

Virtuoso Text Editor User Guide

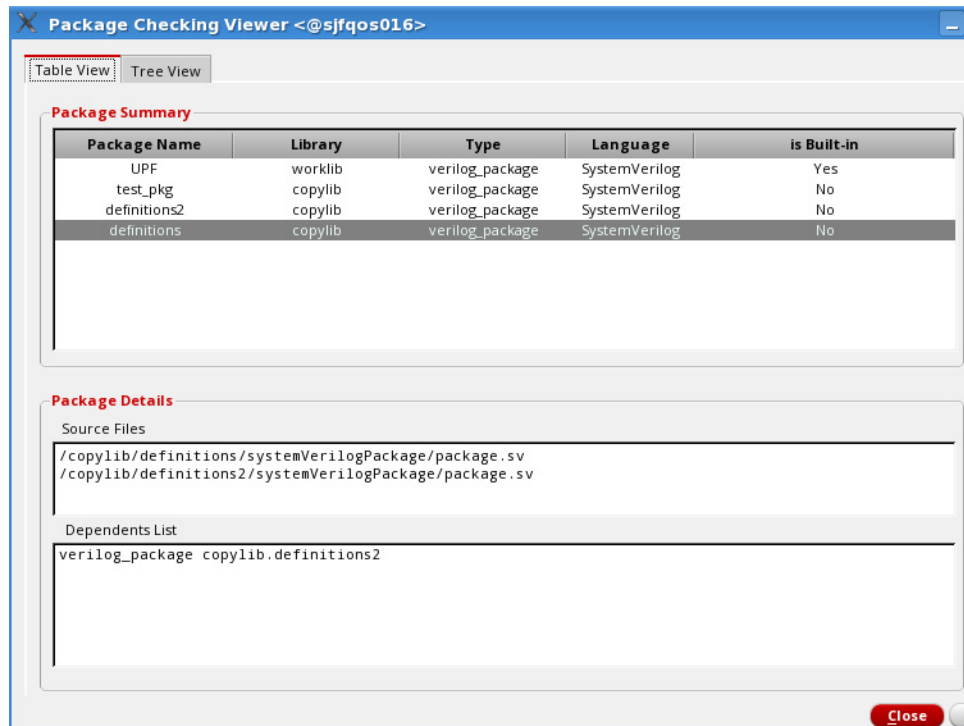
HDL Package Setup

Field Name	Description
makelib table	
<i>On/Off</i>	Enables or disables the use of the HDL package definition in this row.
<i>Library Name</i>	Specifies <i>Library Name</i> .
<i>Source File</i>	Specifies package source files in this table. You can add the <i>makelib source file</i> by clicking <Click here to add package files>.
<i>Options</i>	Specifies <i>-makelib options</i> to add library and file-specific options needed to compile the design units.
	After the local packages are specified, the <i>Edit</i> button is enabled. You can click it to open and edit the file.
<i>Virtuoso cds.lib file</i>	Specifies the <code>cds.lib</code> file.
<i>Check cds.lib</i>	Displays the <code>cds.lib</code> file.
<i>-xmlibdirname</i>	Specifies the name of the <code>xcelium.d</code> directory. Note: The <code>-xmlibdirname</code> is used together with <code>-xmlibdirpath</code> . Therefore, the directory name specified with <code>-xmlibdirname</code> must not contain any path information.
<i>Compile Package Files</i>	Compiles package files based on the setup specified in the Virtuoso HDL Package Setup form, including <code>xrun -f</code> file and the <code>hdl.var</code> file, if specified. The compilation results and <code>xrun.log</code> are generated in <i>Scratch directory</i> . When the compilation is done successfully, the code checks if multiple package files exist, that is, packages by the same name are compiled into different libraries. If the compilation fails, the <code>xrun.log</code> file pop-up opens. You can choose to keep the compilation results or remove them by clicking <i>Clear Compilation Results</i> .
<i>View Log</i>	Displays the <code>xrun.log</code> file saved in the default scratch directory when you click <i>Compile Package Files</i> .

Virtuoso Text Editor User Guide

HDL Package Setup

Field Name	Description
<i>Open Package Checking Viewer</i>	Opens the package checking viewer that has two tabs, table view and tree view.



- Table view: Lists all the package files compiled and found in *Scratch directory*. There are two kinds of package files listed: Cadence Built-In package files that need to be pre-compiled (for example, UVM and UPF) and user-defined package files (which could either be specified in xrunArgs file or in the makelib table). You can view the package details, such as source files and dependents, by selecting a row of a package file. You can view the source file content by double-clicking the source file listed.
Note: For Cadence Built-In package files, the source files are in the Xcelium installation path and are not be displayed.
- Tree view: Lists the compiled module-package dependency relationship, that is, the root node always remains a module and the child nodes are the package files related to the module. If no compiled modules are found or no module-package dependency exists, the tree view is left blank.

Virtuoso Text Editor User Guide

HDL Package Setup

Related Topics

[HDL Package Setup](#)

[Accessing the Virtuoso HDL Package Setup Form](#)

[Use of HDL Package Settings in Various Tools](#)