Virtuoso Text Editor User Guide

Product Version ICADVM20.1
October 2020

Preface

This user guide 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 manual 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`.

This 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 manual assumes that users are familiar with the Cadence SKILL™ language and Virtuoso Verifier.

This preface contains the following topics:

- Scope
- Licensing Requirements
- Related Documentation
- Additional Learning Resources
- Customer Support
- Feedback about Documentation
- Understanding Cadence SKILL
- Typographic and Syntax Conventions

- Identifiers Used to Denote Data Types

Scope

Unless otherwise noted, the functionality described in this guide can be used in both mature node (for example, IC6.1.8) and advanced node and methodologies (for example, ICADVM18.1) release.

Label	Meaning
(ICADVM20.1 Only)	Features supported only in the ICADVM20.1 advanced nodes and advanced methodologies release.
(ICADVM20.1 EXL Only)	Features supported only in the ICADVM20.1 release and which require the Virtuoso_Layout_Suite_EXL license (95800).
(ICADVM20.1 EAD Only)	Features supported only in the ICADVM20.1 release and which require the Virtuoso_Layout_Suite_EAD license (95600).
(IC6.1.8 Only)	Features supported only in mature node releases.
(ICADVM20.1 Virtuoso RF Solution Only)	Features supported only in the ICADVM20.1 release releases and which require the Virtuoso_RF_Option (95560) license.
(ICADVM20.1 Virtuoso MultiTech Framework Only)	Features supported only in the ICADVM20.1 release and which require the Virtuoso_MultiTech_Framework (95022) license.
(ICADVM20.1 Photonics Only)	Features supported only in the ICADVM20.1 release and which require the Virtuoso_Photonics_Option license (95550).

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*.

Related Documentation

What's New and KPNS

- *Virtuoso Text Editor What's New*
- *Virtuoso Text Editor Known Problems and Solutions*

Installation, Environment, and Infrastructure

- *Cadence Installation Guide*
- *Virtuoso Software Licensing and Configuration User Guide*
- *Virtuoso Design Environment User Guide*
- *Cadence Application Infrastructure User Guide*
- *Virtuoso Design Environment SKILL Reference*

Technology Information

- *Virtuoso Technology Data User Guide*
- *Virtuoso Technology Data ASCII Files Reference*
- *Virtuoso Technology Data SKILL Reference*

Virtuoso Tools

- *Virtuoso Schematic Editor User Guide*
- *Virtuoso ADE Assembler User Guide*
- *Virtuoso ADE Explorer User Guide*
- *Virtuoso UltraSim Simulator User Guide*
- *Virtuoso AMS Designer Simulator User Guide*
- *Spectre Circuit Simulator and Accelerated Parallel Simulator RF Analysis in ADE Explorer User Guide*
- *Spectre Circuit Simulator and Accelerated Parallel Simulator User Guide*

- *Spectre Circuit Simulator Reference*

Additional Learning Resources

Video Library

The Video Library on the Cadence Online Support website provides a comprehensive list of videos on various Cadence products.

To view a list of videos related to a specific product, you can use the *Filter Results* feature available in the pane on the left. For example, click the *Virtuoso Layout Suite* product link to view a list of videos available for the product.

You can also save your product preferences in the Product Selection form, which opens when you click the *Edit* icon located next to *My Products*.

Virtuoso Videos Book

You can access certain videos directly from Cadence Help. To learn more about this feature and to access the list of available videos, see Virtuoso Videos.

Rapid Adoption Kits

Cadence provides a number of Rapid Adoption Kits that demonstrate how to use Virtuoso applications in your design flows. These kits contain design databases and instructions on how to run the design flow.

Help and Support Facilities

Virtuoso offers several built-in features to let you access help and support directly from the software.

- The Virtuoso *Help* menu provides consistent help system access across Virtuoso tools and applications. The standard Virtuoso *Help* menu lets you access the most useful help and support resources from the Cadence support and corporate websites directly from the CIW or any Virtuoso application.
- The Virtuoso Welcome Page is a self-help launch pad offering access to a host of useful knowledge resources, including quick links to content available within the Virtuoso installation as well as to other popular online content.

The Welcome Page is displayed by default when you open Cadence Help in standalone mode from a Virtuoso installation. You can also access it at any time by selecting *Help – Virtuoso Documentation Library* from any application window, or by clicking the *Home* button on the Cadence Help toolbar (provided you have not set a custom home page).

For more information, see Getting Help in *Virtuoso Design Environment User Guide.*

Customer Support

For assistance with Cadence products:

- Contact Cadence Customer Support

  Cadence is committed to keeping your design teams productive by providing answers to technical questions and to any queries about the latest software updates and training needs. For more information, visit https://www.cadence.com/support.

- Log on to Cadence Online Support

  Customers with a maintenance contract with Cadence can obtain the latest information about various tools at https://support.cadence.com.

Feedback about Documentation

You can contact Cadence Customer Support to open a service request if you:

- Find erroneous information in a product manual
- Cannot find in a product manual the information you are looking for
- Face an issue while accessing documentation by using Cadence Help

You can also submit feedback by using the following methods:

- In the Cadence Help window, click the *Feedback* button and follow instructions.
- On the Cadence Online Support Product Manuals page, select the required product and submit your feedback by using the *Provide Feedback* box.

Understanding Cadence SKILL

Cadence SKILL is a high-level, interactive programming language based on the popular artificial intelligence language, Lisp. It lets you customize and extend your design environment. Using SKILL you can validate the steps of your algorithm incrementally before incorporating them into a larger program.

For more information about the SKILL language, see Getting Started in the *SKILL Language User Guide*.

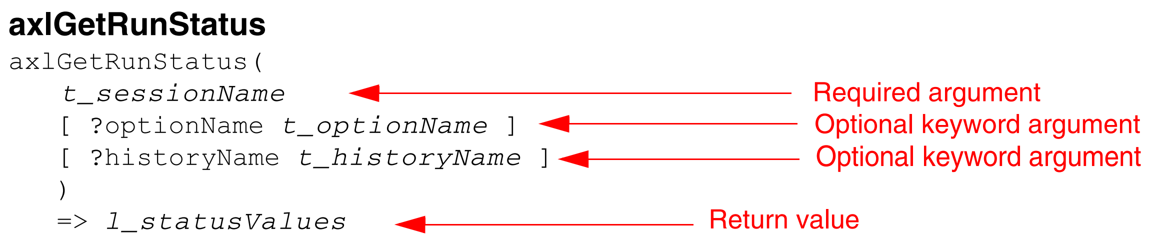
Using SKILL Code Examples

The SKILL APIs in this user manual are explained with illustrative code examples.

You can copy these examples from the manual and paste them directly into the Command Interpreter Window (CIW) or use the code in non-graphical SKILL mode.

Sample SKILL Code

The following code sample shows the syntax of a SKILL API that accepts three arguments.



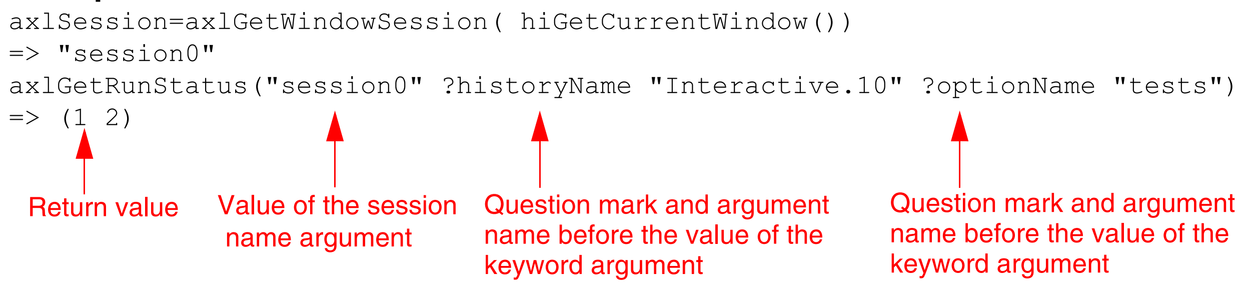
axlGetRunStatus

```
axlGetRunStatus(
    t_sessionName
    [ ?optionName t_optionName ]
    [ ?historyName t_historyName ]
    )
    => l_statusValues
```

The first argument `t_sessionName` is a required argument, where `t` signifies the data type of the argument. The second and third arguments `?optionName t_optionName` and `?historyName t_historyName` are optional keyword arguments (identified by a question mark), which are specified in name-value pairs and can be placed in any order during the function call.

The return value is the value that the SKILL API returns after evaluating the expression. In this case, it is a list of status values, *`l_statusValues`*.

Example



```
axlSession=axlGetWindowSession( hiGetCurrentWindow())
=> "session0"
axlGetRunStatus("session0" ?historyName "Interactive.10" ?optionName "tests")
=> (1 2)
```

Return value
Value of the session name argument
Question mark and argument name before the value of the keyword argument
Question mark and argument name before the value of the keyword argument

Accessing API Help

Quick reference information for SKILL APIs is available from the CIW and the SKILL API Finder. To access the reference information for a particular SKILL API, do one of the following:

- Type `help <function_name>` in the CIW.
- Type `startFinder ( [ ?funcName` *`t_functionName`* `] )` in the CIW.
- Start the SKILL API Finder from the CIW by choosing *Tools – Finder* or type `cdsFinder` on the UNIX command line.

  In the *Search in* field of the displayed Cadence SKILL API Finder window, type the SKILL API name for which you want to display the help information and click *Go*.

  The matches for the searched SKILL API appear in the *Results* area.

  To view the complete documentation of the searched SKILL API, select the API name in the *Results* area and click the *More Info* button. The complete documentation of the selected SKILL API appears in a new Cadence Help window.

Typographic and Syntax Conventions

The following typographic and syntax conventions are used in this manual.

text	Indicates names of manuals, menu commands, buttons, and fields.
`text`	Indicates text that you must type as presented. Typically used to denote command, function, routine, or argument names that must be typed literally.
`z_argument`	Indicates text that you must replace with an appropriate argument value. The prefix (in this example, *`z_`*) indicates the data type the argument can accept and must not be typed.
|	Separates a choice of options.
`{ }`	Encloses a list of choices, separated by vertical bars, from which you **must** choose one.
`[ ]`	Encloses an optional argument or a list of choices separated by vertical bars, from which you **may** choose one.
`[ ?argName` *`t_arg`* `]`	Denotes a *key argument*. The question mark and argument name must be typed as they appear in the syntax and must be followed by the required value for that argument.
`...`	Indicates that you can repeat the previous argument. Used with brackets to indicate that you can specify zero or more arguments. Used without brackets to indicate that you must specify at least one argument.
`,...`	Indicates that multiple arguments must be separated by commas.
`=>`	Indicates the values returned by a Cadence® SKILL® language function.
`/`	Separates the values that can be returned by a Cadence SKILL language function.

If a command-line or SKILL expression is too long to fit within the paragraph margins of this document, the remainder of the expression is moved to the next line and indented. In code excerpts, a backslash (\ ) indicates that the current line continues on to the next line.

Identifiers Used to Denote Data Types

Data type identifiers are used to indicate the type of value required by an API argument. These data types are denoted by a single letter that is prefixed to the argument label and is separated from the argument by an underscore; for example, `t` is the data type in *`t_viewName`*. Data types and underscores are used only as identifiers; they must not be typed when specifying the argument in a function.

Prefix	Internal Name	Data Type
a	array	array
A	amsobject	AMS object
b	ddUserType	DDPI object
B	ddCatUserType	DDPI category object
C	opfcontext	OPF context
d	dbobject	Cadence database object (CDBA)
e	envobj	environment
f	flonum	floating-point number
F	opffile	OPF file ID
g	general	any data type
G	gdmSpecIlUserType	generic design management (GDM) spec object
h	hdbobject	hierarchical database configuration object
I	dbgenobject	CDB generator object
K	mapiobject	MAPI object
l	list	linked list
L	tc	Technology file time stamp
m	nmpIlUserType	nmpIl user type
M	cdsEvalObject	cdsEvalObject
n	number	integer or floating-point number
o	userType	user-defined type (other)
p	port	I/O port
q	gdmspecListIlUserType	gdm spec list

Prefix	**Internal Name**	**Data Type**
r	defstruct	defstruct
R	rodObj	relative object design (ROD) object
s	symbol	symbol
S	stringSymbol	symbol or character string
t	string	character string (text)
T	txobject	transient object
u	function	function object, either the name of a function (symbol) or a lambda function body (list)
U	funobj	function object
v	hdbpath	hdbpath
w	wtype	window type
sw	swtype	subtype session window
dw	dwtype	subtype dockable window
x	integer	integer number
y	binary	binary function
&	pointer	pointer type

For more information, see *Cadence SKILL Language User Guide*.

1

Introduction

This chapter includes the following topics:

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.

Note: 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	**1.** Choose *File — Open*. The Open File form appears. **2.** Select the text cellview and click *OK*. For details, see *Working with Cellviews* in *Virtuoso Design Environment User Guide*.

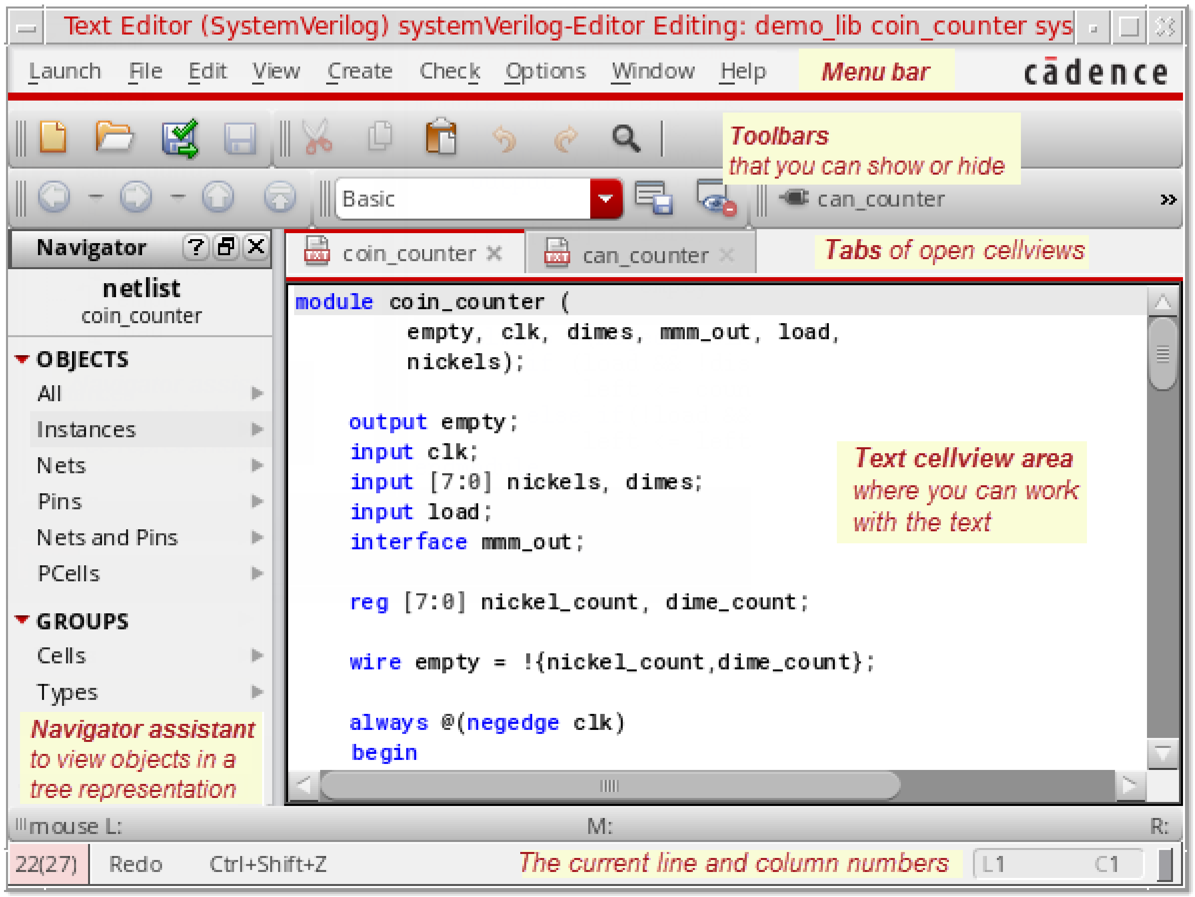
Virtuoso Application	**Process to Launch Virtuoso Text Editor**
Virtuoso Schematic Editor	**1.** Open a design and double-click the symbol associated with a text cellview. The Descend form appears. **2.** Select the text cellview and click *OK*. For details, see *Descending Using the Descend Command* in *Virtuoso Schematic Editor L User Guide*.

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	**1.** Choose *File — New — Cell View*. The New File form appears. **2.** Specify the library, cell, and view. Select one of the HDL languages from the *Type* list. **3.** Click *OK*.
Virtuoso CIW	**1.** Choose *File — New — Cellview*. The New File form appears. **2.** Specify the library, cell, and view. Select one of the HDL languages from the *Type* list. **3.** Click *OK*.
Virtuoso Schematic Editor	**1.** Choose *File — New*. The New File form appears. **2.** Specify the library, cell, and view. Select one of the HDL languages from the *Type* list. **3.** Click *OK*.

Understanding the Graphical User Interface

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


Text Editor (SystemVerilog) systemVerilog-Editor Editing: demo_lib coin_counter sys
Launch File Edit View Create Check Options Window Help
Menu bar
cādence
Toolbars
that you can show or hide
Basic
can_counter
Navigator
netlist
coin_counter
OBJECTS
All
Instances
Nets
Pins
Nets and Pins
PCells
GROUPS
Cells
Types
Navigator assistant to view objects in a tree representation
coin_counter
can_counter
Tabs of open cellviews
Text cellview area where you can work with the text
mouse L:
M:
R:
22(27)
Redo
Ctrl+Shift+Z
The current line and column numbers
L1
C1

```
module coin_counter (
        empty, clk, dimes, mmm_out, load,
        nickels);

    output empty;
    input clk;
    input [7:0] nickels, dimes;
    input load;
    interface mmm_out;

    reg [7:0] nickel_count, dime_count;

    wire empty = !{nickel_count,dime_count};

    always @(negedge clk)
    begin
```

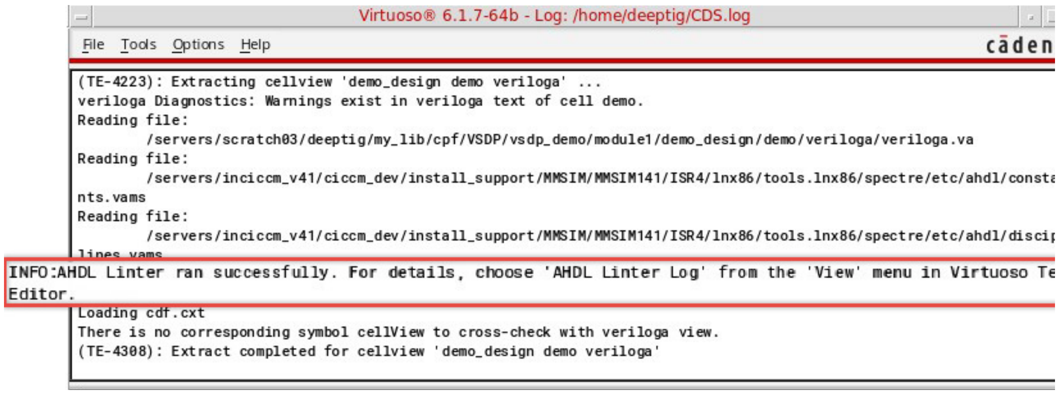
The title bar of Virtuoso Text Editor indicates the library, cell, and text cellview opened in the editor, along with the read or write mode and the language of the text cellview.

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

UI Component	Description
Text cellview area	Displays the contents of the text cellview. 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.
Navigator assistant	Provides facilities to view objects across the design hierarchy using a tree representation. For details, see *The Navigator Assistant* in *Virtuoso Schematic Editor L User Guide*. Also see “Navigating a Design Hierarchy Containing Text Cellviews” on page 45.
Menu bar	
Launch menu	Provides the option to launch the NC-Verilog Integration Environment, if a Verilog text cellview is opened. For details on this environment, see *Virtuoso NC-Verilog Environment User Guide*.

UI Component	Description
File menu	■ *New* – Create a new cellview. ■ *Open* – Open an existing cellview. ■ *Close* – Close the displayed text cellview. ■ *Save* – Save the text cellview without checking for syntax errors. ■ *Check* – Check the syntax, save, and parse the text cellview. ■ *Extract* – 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. ■ *Make Editable/Read-Only* – Switch text cellview mode to read-only or edit. ■ *Discard Edits* – Discard unsaved edits and reload the text cellview. ■ *Reload* – 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. ■ *Print* – Print the contents of the text cellview file. ■ *Bookmarks* – Add and manage bookmarks. For details, see *Using Bookmarks and Views* in the *Virtuoso Design Environment User Guide*. ■ Cellview list – List the previously opened cellviews for quick access. ■ *Close All* – 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.

UI Component	Description
Edit menu	■ *Undo, Redo* – Undo or redo changes. ■ *Copy, Cut, Paste* – Copy, cut, and paste content. ■ *Find, Find Previous, Find and Replace* – Find and replace text. ■ *Go to Line* – Go to a specific line number. ■ *Hierarchy* – Navigate the design hierarchy. ■ *Launch External Editor* – Open the text cellview in an external editor. ■ *Select All* – Select all text in the work area.
View menu	■ *Show Line Numbers* – Show or hide the line numbers. ■ *Parser Log File* – View the text cellview file parser log.
Create menu	■ *Cellview From Cellview* – Create a new cellview from an existing cellview. For details, see “Creating a Cellview from an Existing Cellview” on page 32 and *Automatically Creating a Cellview from another Cellview* in *Virtuoso Schematic Editor L User Guide*.
Check menu	■ *Pin Order* – Check the pin order. This feature is useful for text cellview files where terminals of instances are connected by sequence. ■ *Remove All Markers* – Remove all displayed markers, such as syntax error markers.

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. 

```
(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/consta
nts.vams
Reading file:
        /servers/inciccm_v41/ciccm_dev/install_support/MMSIM/MMSIM141/ISR4/lnx86/tools.lnx86/spectre/etc/ahdl/discip
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'
```

UI Component	Description
Window menu	■ *Assistants* – Access the Navigator assistant. For details, see “Navigating a Design Hierarchy Containing Text Cellviews” on page 45. ■ *Toolbars* – View or hide the toolbars and customize the *File* and *Edit* toolbars using Toolbar Manager. For details, see *Using Toolbar Manager* in *Virtuoso Design Environment User Guide*. The available toolbars are *File*, *Edit*, *Bookmarks*, *Go*, and *Workspaces*. ■ *Workspaces* – Set the workspace. For details, see *Working with Workspaces* in *Virtuoso Design Environment User Guide*. ■ *Tabs* – Navigate between the opened tabs. You can also close the current or other opened tabs. ■ *Copy Window* – Copy the current window. Any changes you make in the opened text cellview file is reflected in all the copies of the window.
Help menu	■ Access help and online support.
Toolbars	
Note: You can customize the *File* and *Edit* toolbars. For details, see *Using Toolbar Manager* in *Virtuoso Design Environment User Guide*.	
File toolbar	■ 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.
Edit toolbar	■ Cut, copy, paste content. ■ Undo and redo changes. ■ Search the text cellview content.
Bookmark toolbar	■ Access and manage bookmarks. For details, see *Using Bookmarks and Views* in *Virtuoso Design Environment User Guide*.

UI Component	Description
Go toolbar	■ Navigate the design hierarchy. For details, see *Navigating Cellviews and Hierarchies* in *Virtuoso Design Environment User Guide*.
Workspaces toolbar	■ Set and optimize the workspace. For details, see *Working with Workspaces* in *Virtuoso Design Environment User Guide*.

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.

2

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

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*

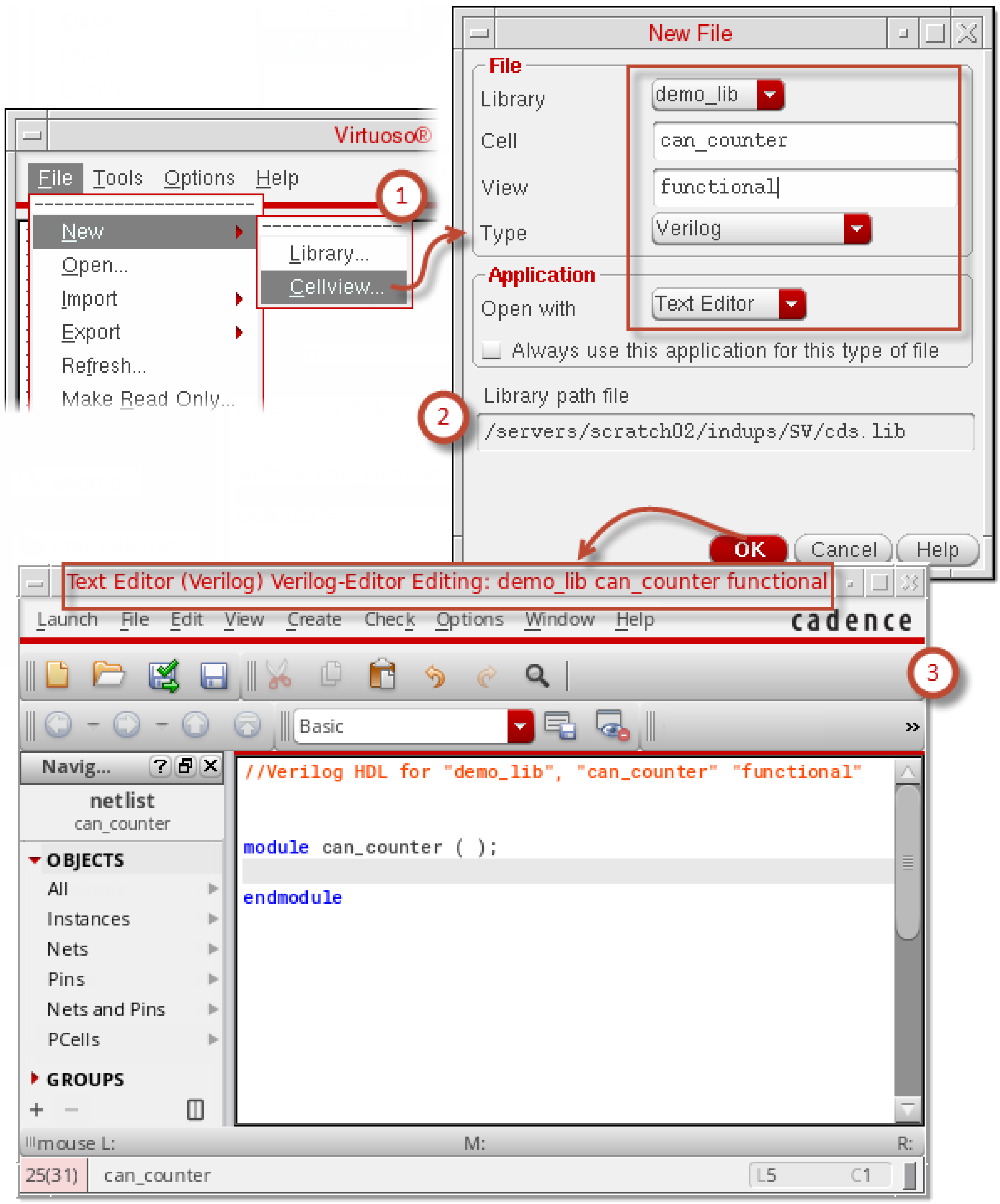
You can also use the command `cdsTextTo5x` to import Verilog, SystemVerilog, and VHDL text files into the Virtuoso design environment. For details, see Importing Design Data by Using cdsTextTo5x.

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

Creating a Text Cellview

There are several ways to initiate the creation of a new text cellview, as described in "Launching Virtuoso Text Editor" on page 18. The following figure illustrates how you initiate the text cellview creation from Virtuoso Library Manager.



Library: demo_lib
Cell: can_counter
View: functional
Type: Verilog
Open with: Text Editor
Library path file: /servers/scratch02/indups/SV/cds.lib

Text Editor (Verilog) Verilog-Editor Editing: demo_lib can_counter functional

```
//Verilog HDL for "demo_lib", "can_counter" "functional"


module can_counter ( );

endmodule
```

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” on page 18.

   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.

Note: 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

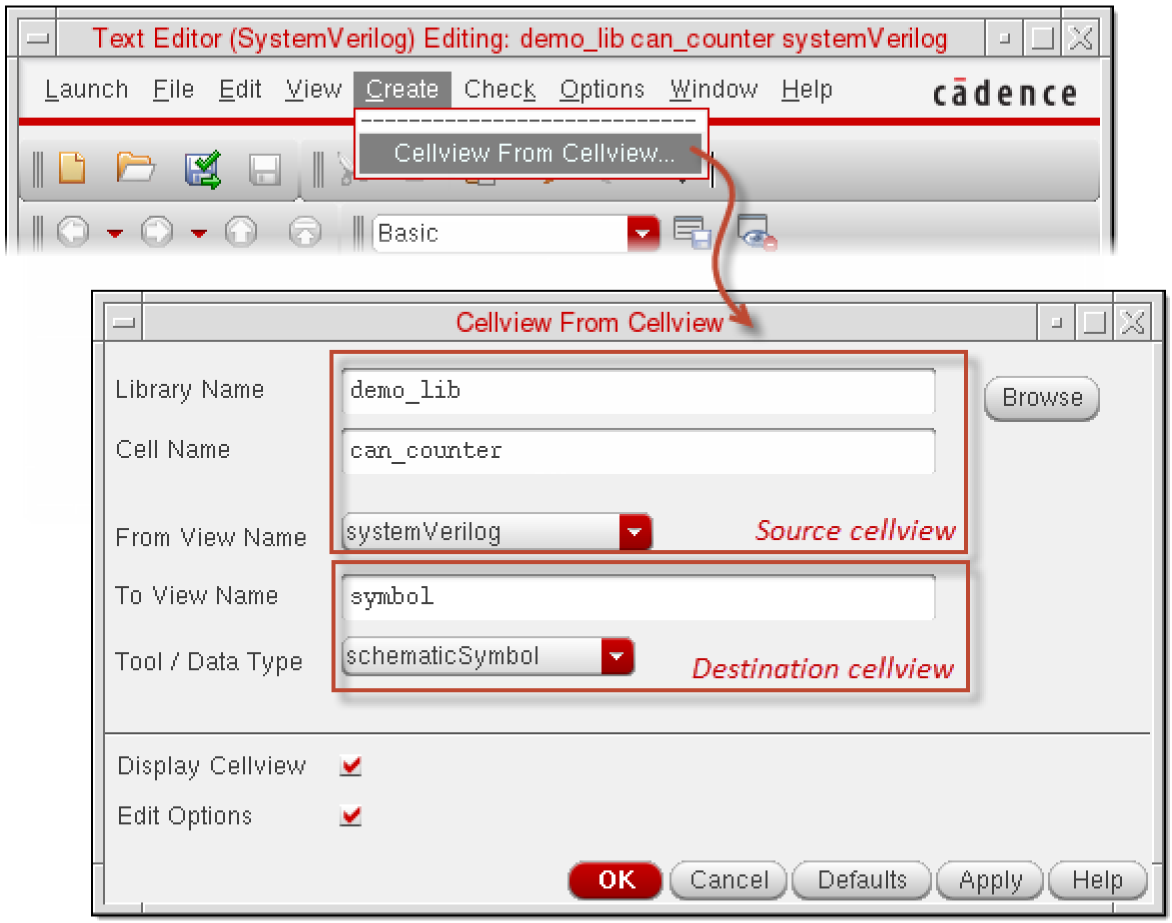
Using Virtuoso Text Editor, you can create a cellview using another cellview as a source. For example, you can create a symbol cellview from a Verilog text cellview.

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.

Notes:

- 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*.

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


Text Editor (SystemVerilog) Editing: demo_lib can_counter systemVerilog
Launch File Edit View Create Check Options Window Help
cādence
Cellview From Cellview...
Basic
Cellview From Cellview
Browse
Source cellview
Destination cellview
OK
Cancel
Defaults
Apply
Help

Library Name	demo_lib
Cell Name	can_counter
From View Name	systemVerilog
To View Name	symbol
Tool / Data Type	schematicSymbol
Display Cellview	✓
Edit Options	✓

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	

Field	Description
Library Name	Specify the library name of the source cellview.
Cell Name	Specify the cell name of the source cellview.
From the View Name	Select the source view name.
Destination cellview	
To View Name	Specify the destination view name.
Tool / Data Type	Select the destination view type. The *To View Name* field gets updated with the default view name of the selected type.
Options	
Display Cellview	Select to open the new cellview in a new window.
Edit Options	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:

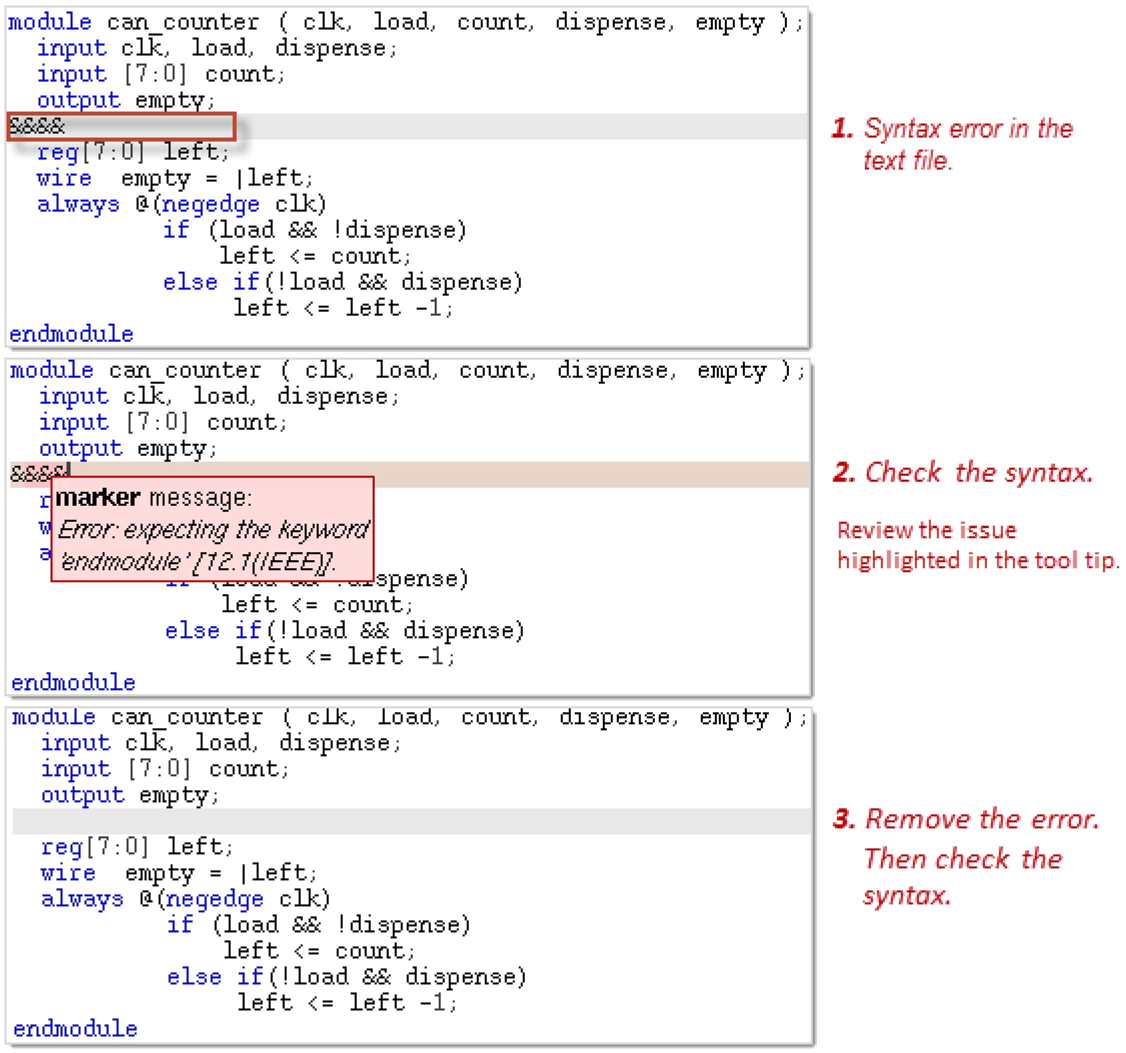
➡ Choose *File — Make Editable*, or *File — Read-Only*. The available option depends on the current mode.

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.



```
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
```

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;
&&&&
```

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

```
               left <= count;
           else if(!load && dispense)
                left <= left -1;
endmodule
```

2. *Check the syntax.*

Review the issue highlighted in the tool tip.

```
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.*

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 41.
- 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.

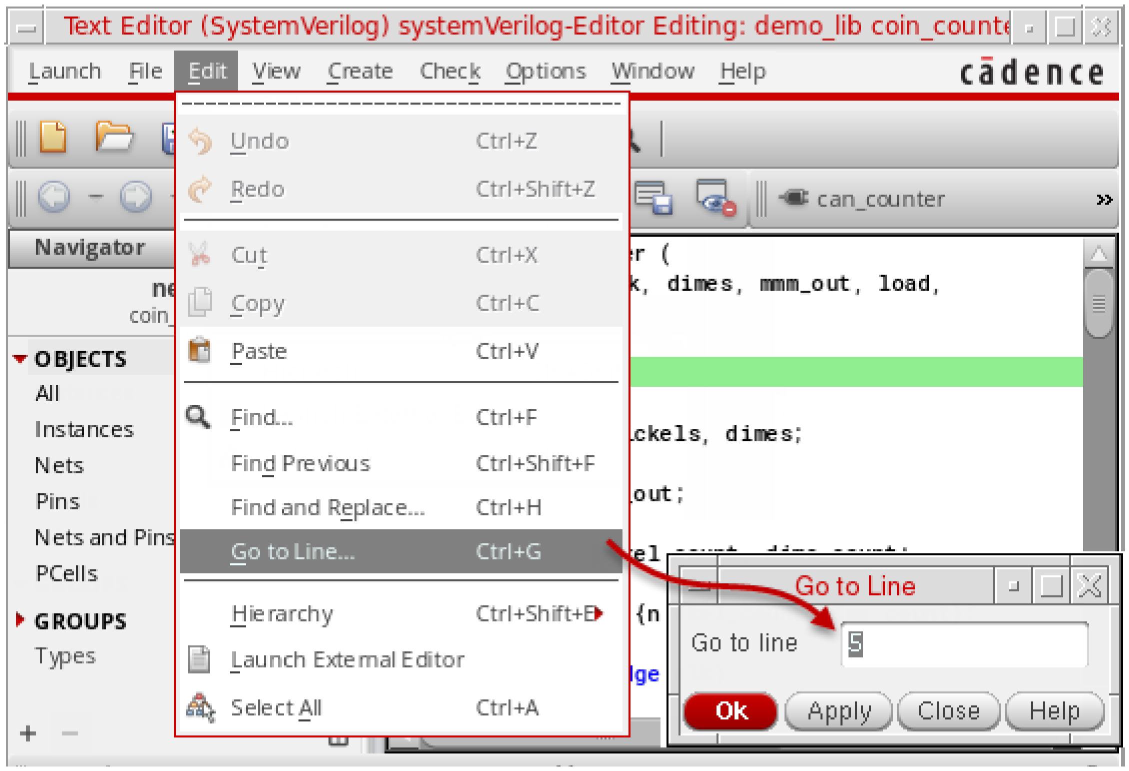
To show or hide the line numbers:

➡ 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 66.

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 begining 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.

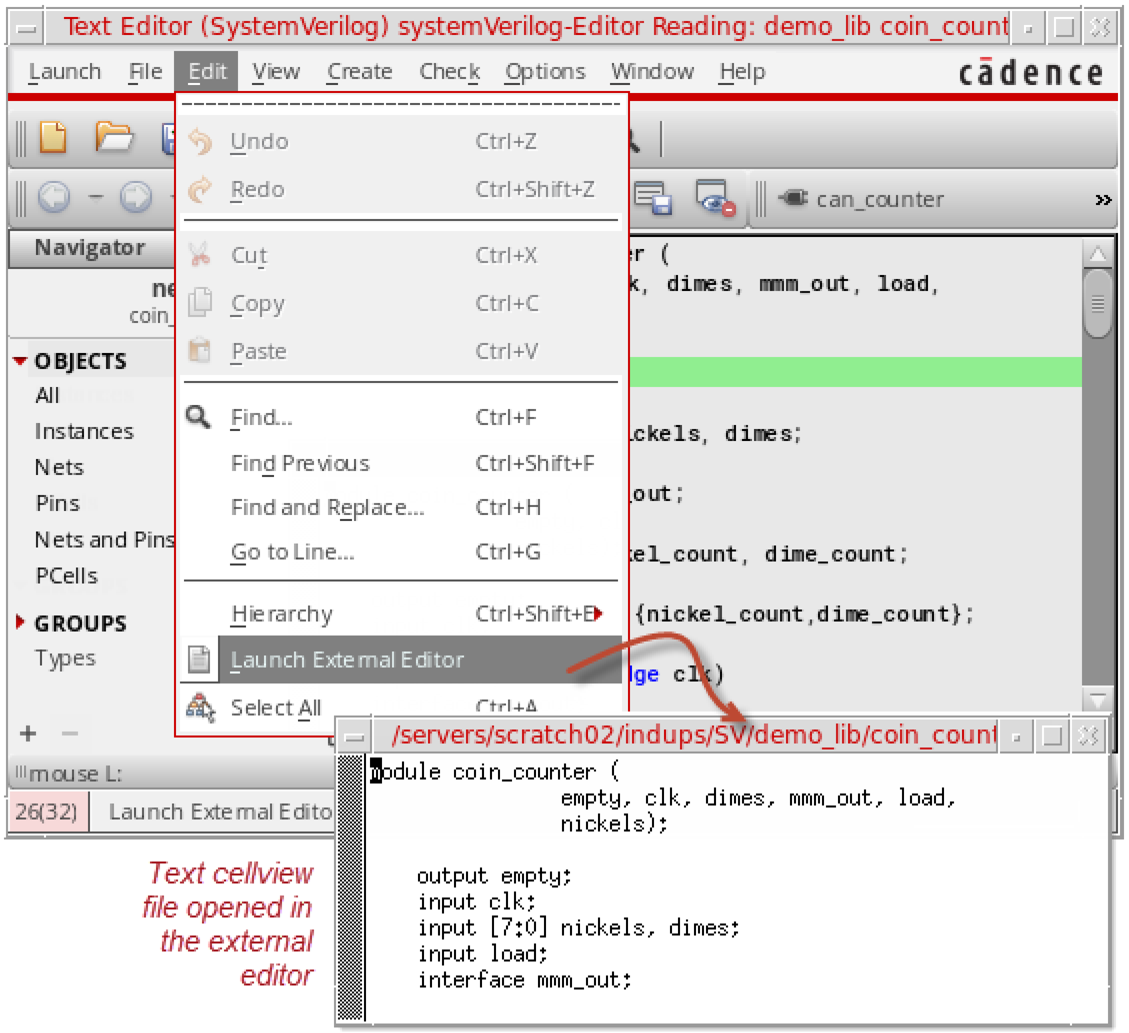
Editing a Text Cellview File in an External Editor

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.

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


Text Editor (SystemVerilog) systemVerilog-Editor Reading: demo_lib coin_count
Launch File Edit View Create Check Options Window Help
cādence
Undo Ctrl+Z
Redo Ctrl+Shift+Z
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Find... Ctrl+F
Find Previous Ctrl+Shift+F
Find and Replace... Ctrl+H
Go to Line... Ctrl+G
Hierarchy Ctrl+Shift+E
Launch External Editor
Select All Ctrl+A
can_counter
Navigator
OBJECTS
All
Instances
Nets
Pins
Nets and Pins
PCells
GROUPS
Types
mouse L:
26(32)
Launch External Edito
/servers/scratch02/indups/SV/demo_lib/coin_count
Text cellview file opened in the external editor

```
module coin_counter (
                empty, clk, dimes, mmm_out, load,
                nickels);

    output empty;
    input clk;
    input [7:0] nickels, dimes;
    input load;
    interface mmm_out;
```

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 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.

Note: You can configure the behavior of the port check functionality using the environment variables `disablePortOrderPopup` and `disablePortOrderCheck`. For details, see *Resolving Pin Mismatch* in *Virtuoso Schematic Editor L User Guide*.

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 20.

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

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 43.

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 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
gpdk090_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

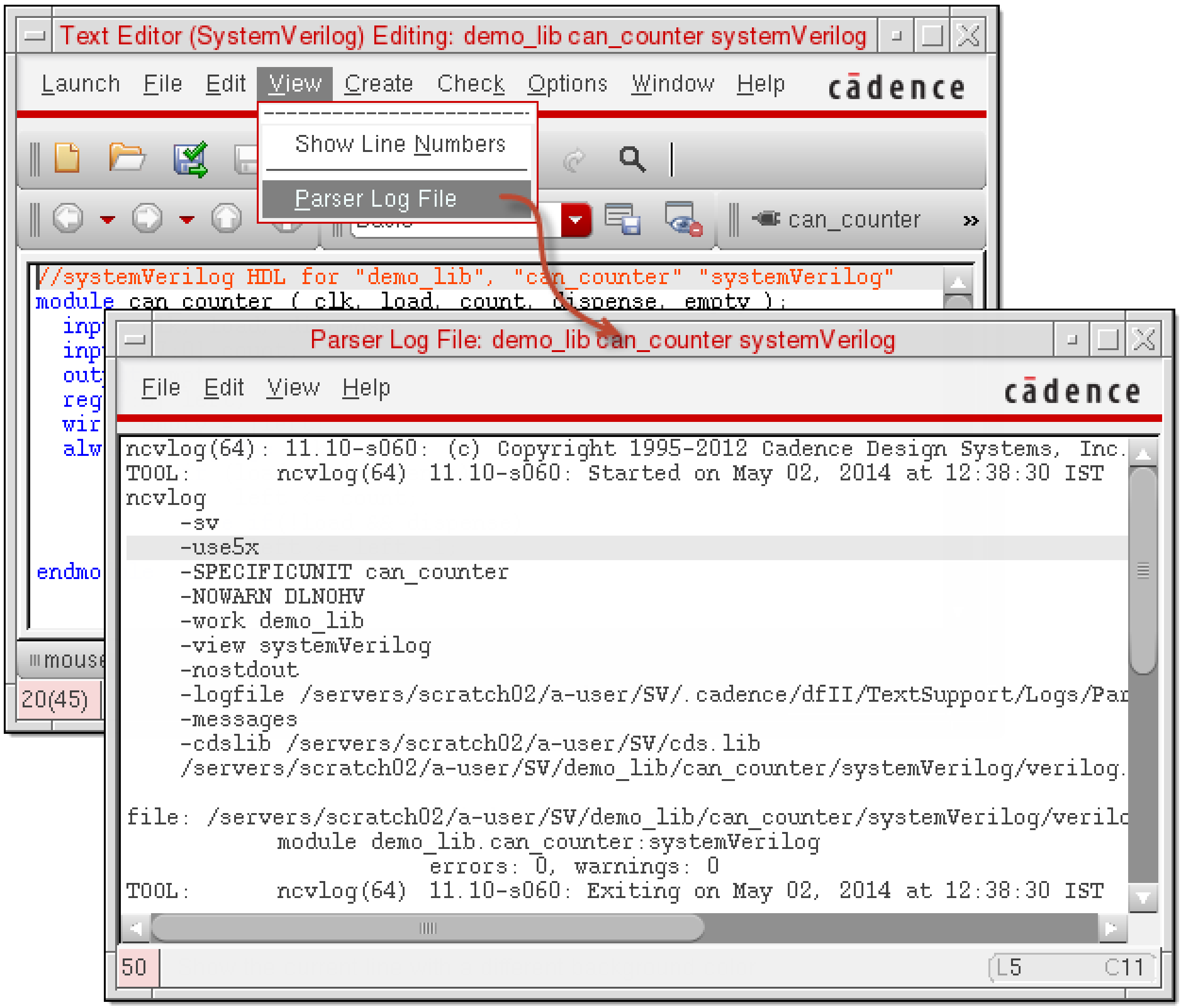
Virtuoso Text Editor uses the appropriate parser to parse the file of a text cellview to perform operations, such as generating the cellview database. When the parser processes a file, it maintains a log. You can view this log for details. For example, you can view the log to investigate parse errors.

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.

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


Text Editor (SystemVerilog) Editing: demo_lib can_counter systemVerilog
Launch File Edit View Create Check Options Window Help
cādence
Show Line Numbers
Parser Log File
can_counter
Parser Log File: demo_lib can_counter systemVerilog
File Edit View Help
20(45)
50
L5 C11

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

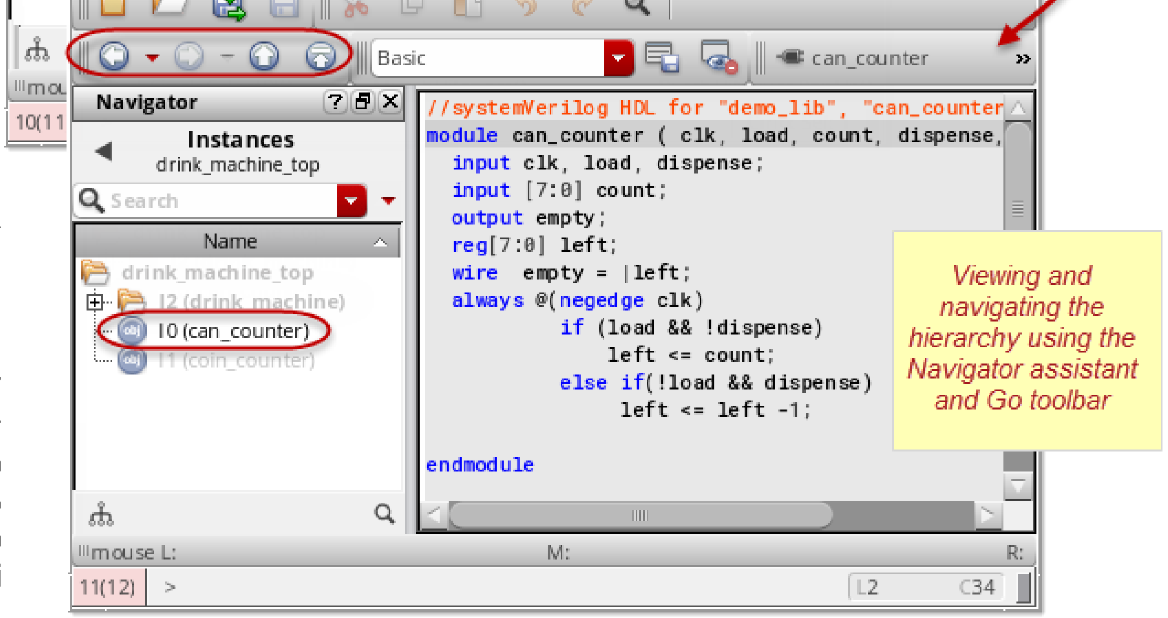
```
ncvlog(64): 11.10-s060: (c) Copyright 1995-2012 Cadence Design Systems, Inc.
TOOL:       ncvlog(64)  11.10-s060: Started on May 02, 2014 at 12:38:30 IST
ncvlog
        -sv
        -use5x
        -SPECIFICUNIT can_counter
        -NOWARN DLNOHV
        -work demo_lib
        -view systemVerilog
        -nostdout
        -logfile /servers/scratch02/a-user/SV/.cadence/dfII/TextSupport/Logs/Par
        -messages
        -cdslib /servers/scratch02/a-user/SV/cds.lib
        /servers/scratch02/a-user/SV/demo_lib/can_counter/systemVerilog/verilog.

file: /servers/scratch02/a-user/SV/demo_lib/can_counter/systemVerilog/verilo
            module demo_lib.can_counter:systemVerilog
                        errors: 0, warnings: 0
TOOL:       ncvlog(64)  11.10-s060: Exiting on May 02, 2014 at 12:38:30 IST
```

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.



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

Viewing and navigating the hierarchy using the Navigator assistant and Go toolbar

iews

cend to
igate a
xt
er

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*.

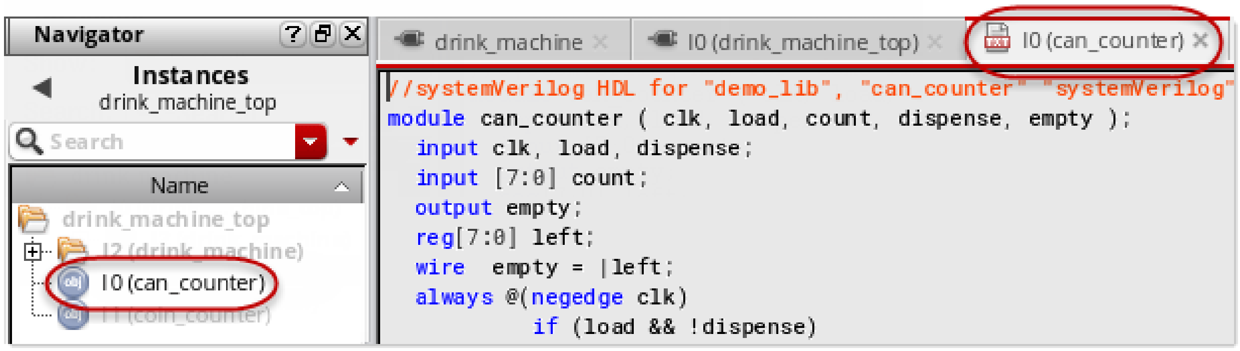
Tip

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*.

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.

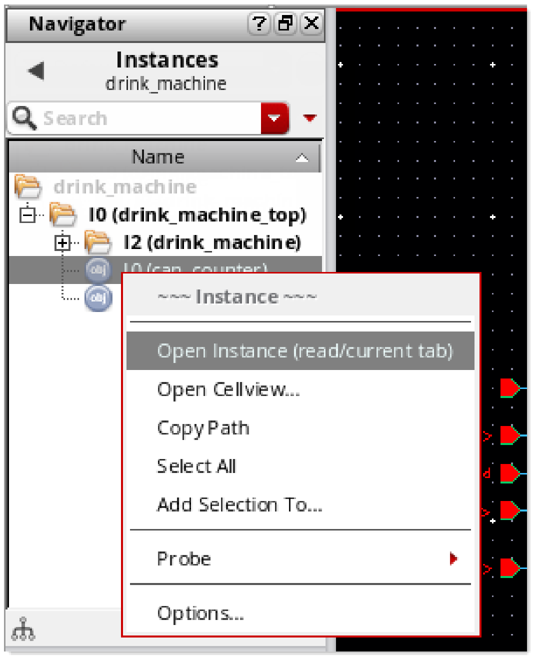


```
//systemVerilog HDL for "demo_lib", "can_counter" "systemVerilog"
module can_counter ( clk, load, count, dispense, empty );
  input clk, load, dispense;
  input [7:0] count;
  output empty;
  reg[7:0] left;
  wire  empty = |left;
  always @(negedge clk)
          if (load && !dispense)
```

- 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.



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.

A

Virtuoso Text Editor Functions

You can use the following SKILL functions in Virtuoso Text Editor:

- teDiscardEdits
- teGetCursorPosition
- teIsModified
- teSave
- teSaveWithDerivedData
- teSetCursor
- teSetEditMode
- teRegPostExtractTrigger
- teRegPreExtractTrigger
- teUnRegPostExtractTrigger
- teUnRegPreExtractTrigger
- hdlGenerateTextDatabase

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

d_cvId	Cellview ID of the schematic in which the edits have been discarded.

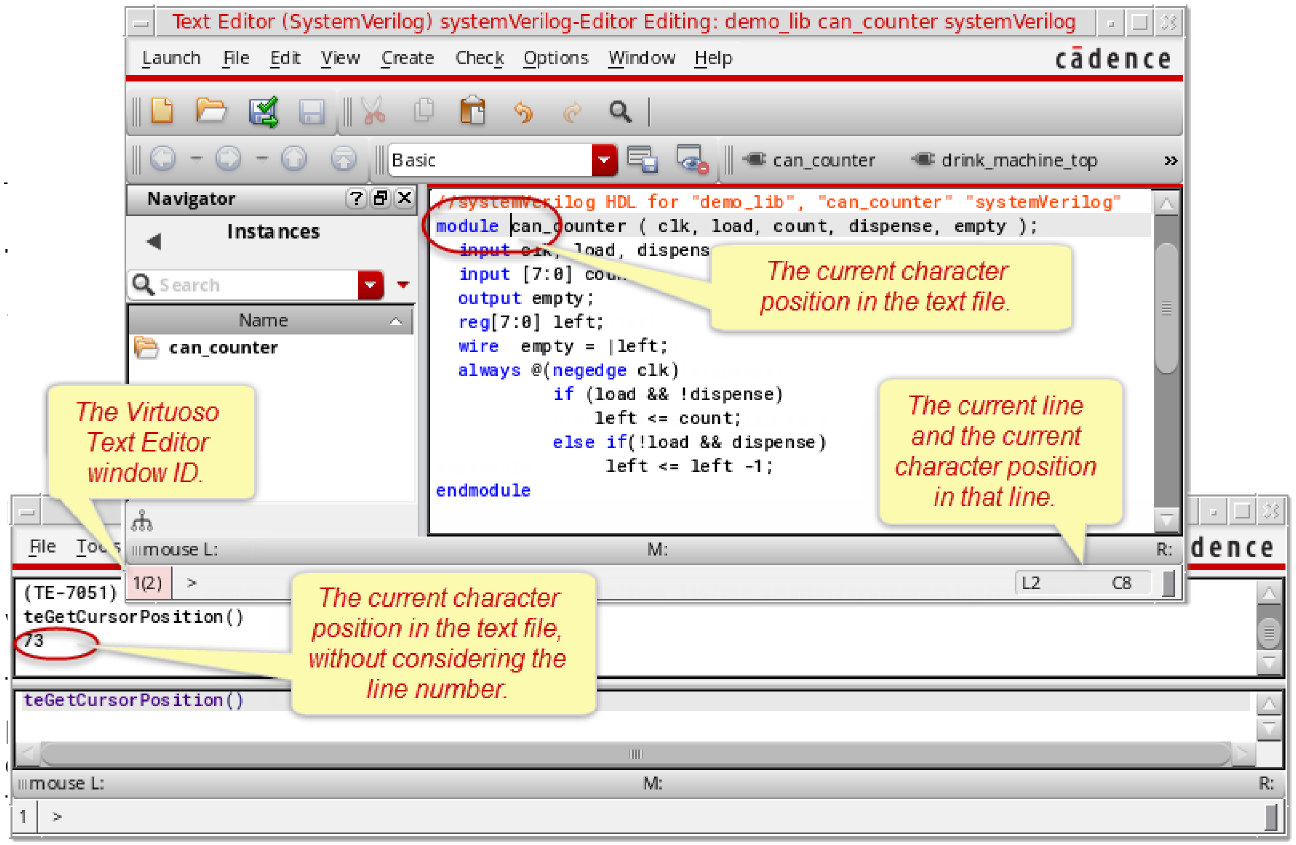
Value Returned

`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)
```


Text Editor (SystemVerilog) systemVerilog-Editor Editing: demo_lib can_counter systemVerilog
Launch File Edit View Create Check Options Window Help
cādence
Basic
can_counter
drink_machine_top
Navigator
Instances
Search
Name
can_counter
The current character position in the text file.
The Virtuoso Text Editor window ID.
The current line and the current character position in that line.
The current character position in the text file, without considering the line number.
mouse L:
M:
R:
1(2)
L2
C8
1

```
//systemVerilog HDL for "demo_lib", "can_counter" "systemVerilog"
module can_counter ( clk, load, count, dispense, empty );
  input clk, load, dispens
  input [7:0] cou
  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
```

```
(TE-7051)
teGetCursorPosition()
73
```

```
teGetCursorPosition()
```

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

d_cvId	Cellview ID of the schematic which is being checked.

Value Returned

`t`	The text cellview has been modified by any active text editor.
`nil`	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

d_cvId	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

d_cvId	Cellview ID of the schematic which is being saved.

Value Returned

`t`	The text cellview has been saved and derived data created successfully.
`nil`	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

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

Value Returned

`t`	The cursor is placed at the specified character position.
`nil`	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 begining 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 specifed cellview to read-only(`r`)or editable(`a`) based on the value of the mode argument.

Arguments

`d_cvId`	Cellview ID of the schematic which is being set to the edit mode.
`d_mode`	The editing mode.

Value Returned

`t`	The edit mode of the text cellview changed successfully.
`nil`	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 to be triggered after the text editor extract functionality is being run.

Arguments

s_symbol	SKILL symbol for the trigger function.

Value Returned

`t`	The trigger registered successfully.
`nil`	The trigger could not register.

Example

You can register `postTrigger` as shown here:

```
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)
   t
)

teRegPostExtractTrigger('postTrigger)
```

teRegPreExtractTrigger

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

Description

Registers a user-defined function to be triggered prior to the text editor extract functionality is being run.

Arguments

s_symbol	SKILL symbol for the trigger function.

Value Returned

`t`	The trigger registered successfully.
`nil`	The trigger could not register.

Example

You can register `preTrigger` as shown here:

```
defun( preTrigger (cvId)
  info("*** preTrigger called with lcv = '%s %s %s'\n"
        cvId~>parent~>parent~>name cvId~>parent~>name cvId~>name)
  t
)

teRegPreExtractTrigger('preTrigger)
```

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

s_symbol	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

s_symbol	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

t_libName	Specifies the library name of the cellview.
t_cellName	Specifies the name of the cellview.
t_viewName	Specifies the view name of the cellview.

Value Returned

`t`	If the database is generated successfully.
`nil`	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
    )
)
```

B

Environment Variables

This appendix describes the environment variables of Virtuoso Text Editor 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 environment variables:

- inScopeReadBGColor on page 66
- inScopeWriteBGColor on page 66
- showLineNumbersInSideBar on page 66
- useExternalEditor on page 67
- enableAhdllintStaticCheck on page 68
- syntaxLineLength on page 69
- CDS5X_NOLINK on page 67

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.

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 topic: “Switching Between Edit Mode and Read-Only Mode” on page 35

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 topic: “Switching Between Edit Mode and Read-Only Mode” on page 35

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	`t`
Acceptable Values	`t`, `nil`

Related topic: “Showing Line Numbers and Going to a Line Number” on page 37

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	`nil`
Acceptable Values	`t`, `nil`

Related topic: "Editing a Text Cellview File in an External Editor" on page 39

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	`t`
Acceptable Values	`t`, `nil`

enableAhdllintStaticCheck

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

Variable Type	Boolean
Default Value	`nil`
Acceptable Values	`t`, `nil`

Related topic: “Understanding the Graphical User Interface” on page 20

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`

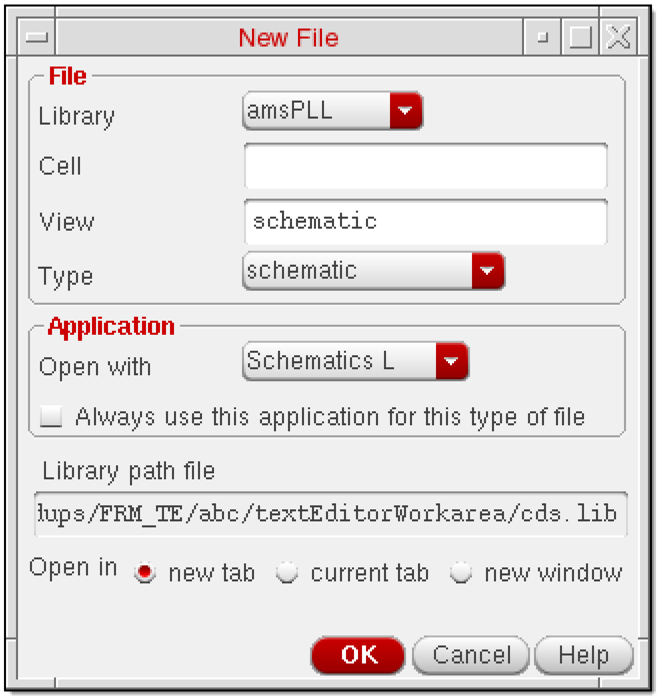
C

Virtuoso Text Editor Forms

This appendix describes the following Virtuoso Text Editor forms:

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.



File
Library: amsPLL
Cell:
View: schematic
Type: schematic

Application
Open with: Schematics L
Always use this application for this type of file [ ]

Library path file: lups/FRM_TE/abc/textEditorWorkarea/cds.lib

Open in: [✓ new tab] current tab new window

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

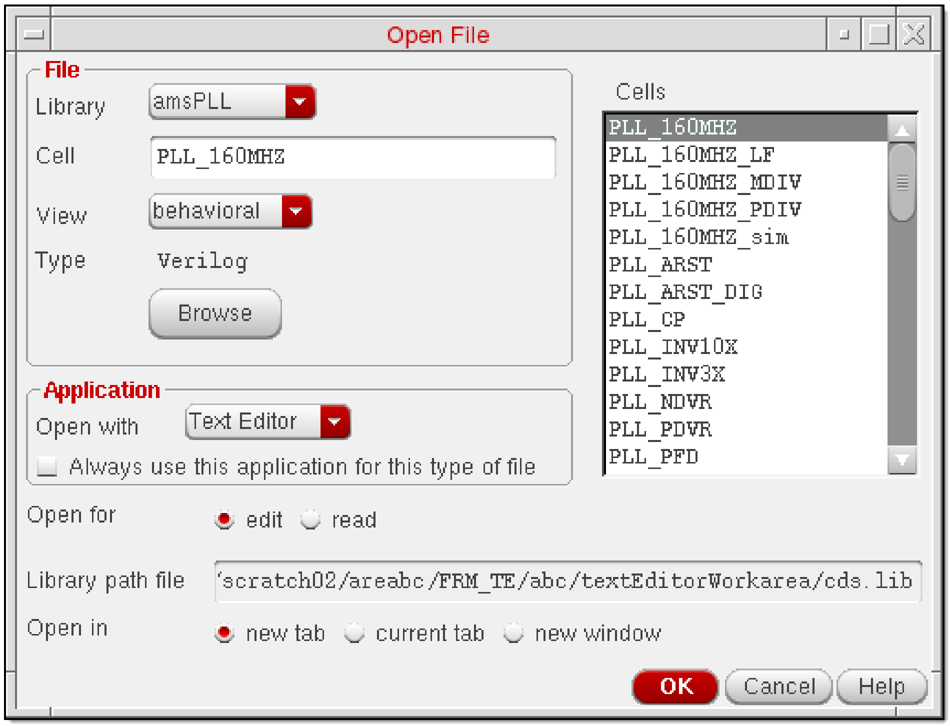
Form Component	Description
Always use this application for this type of file	Select to make the chosen application as the default application to open the specified type of cellview.
Open in	Select the location where you want to open the new cellview. You can open the cellview in a new or same tab in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

References:

- “Creating a Text Cellview” on page 29
- *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.



Library: amsPLL
Cell: PLL_160MHZ
View: behavioral
Type: Verilog
Open with: Text Editor
☐ Always use this application for this type of file
Open for: [✓ edit] read
Library path file: 'scratch02/areabc/FRM_TE/abc/textEditorWorkarea/cds.lib
Open in: [✓ new tab] current tab new window

Cells:
PLL_160MHZ
PLL_160MHZ_LF
PLL_160MHZ_MDIV
PLL_160MHZ_PDIV
PLL_160MHZ_sim
PLL_ARST
PLL_ARST_DIG
PLL_CP
PLL_INV10X
PLL_INV3X
PLL_NDVR
PLL_PDVR
PLL_PFD

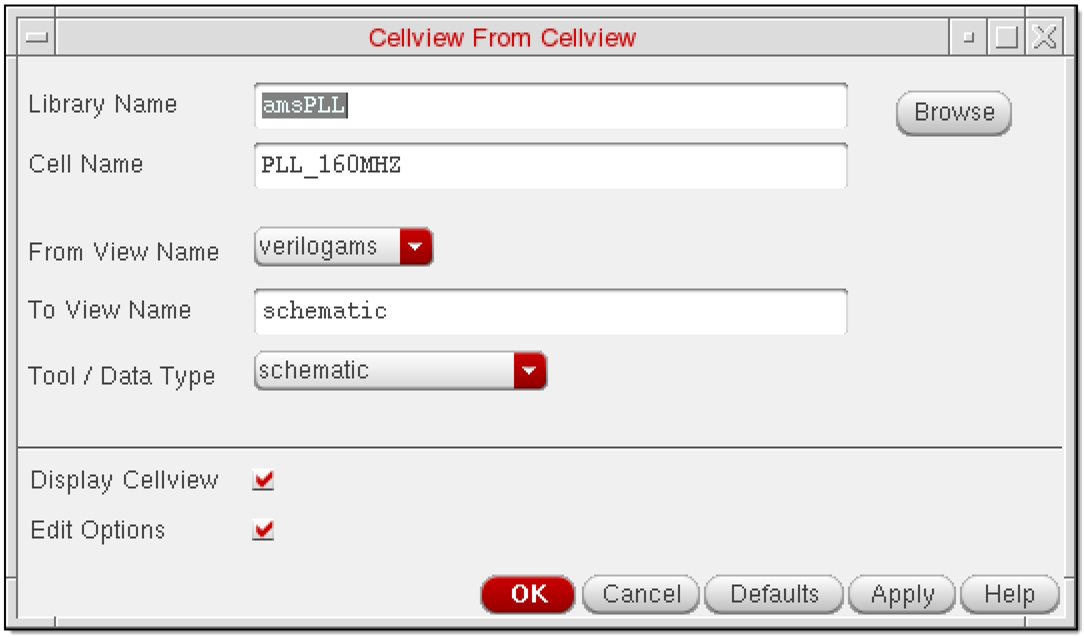
Form Component	**Description**
Library	Choose the name of the library that contains the cellview you want to open. The *Cells* area updates to display the cells available in the library.
Cell	Type the cell name or select the name from the *Cells* area.
View	Choose one of the available views of the cell you want to open. The *Type* field updates to display the type of the selected view.
Browse	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.
Open with	Choose the application that is invoked to display the new cellview.

Form Component	Description
Always use this application for this type of file	Select to make the chosen application as the default application to open the specified type of cellview.
Open for	Select *edit* to open the cellview in edit mode, or *read* to open it in read-only mode.
Open in	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 18

Cellview From Cellview Form

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


Cellview From Cellview

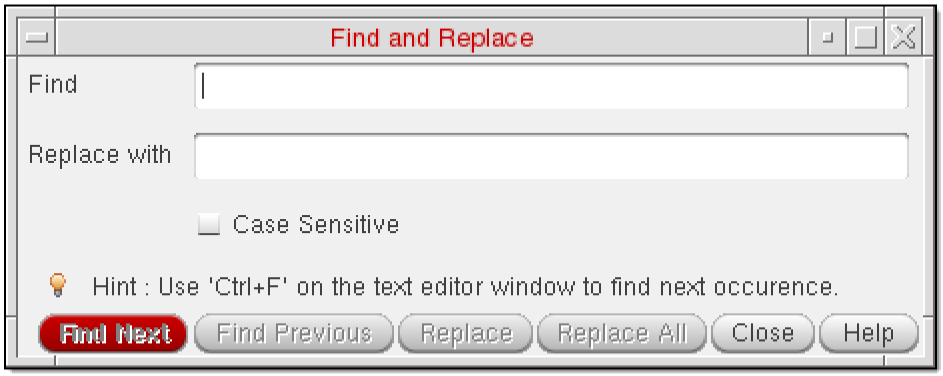
Library Name	amsPLL
Cell Name	PLL_160MHZ
From View Name	verilogams
To View Name	schematic
Tool / Data Type	schematic
Display Cellview	✓
Edit Options	✓

Form Component	Description
Library Name	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 *Browse* and choose the library, cell, and view from Library Browser.
Cell Name	Specify the cell name of the source cellview.
From the View Name	Select the source view name.
To View Name	Specify the destination view name.
Tool / Data Type	Select the destination view type. The *To View Name* field updates with the default view name of the selected type.
Display Cellview	Select to open the new cellview in a new window.
Edit Options	Select to edit any additional options before the cellview creation.

Reference: "Creating a Cellview from an Existing Cellview" on page 32

Find and Replace Form

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



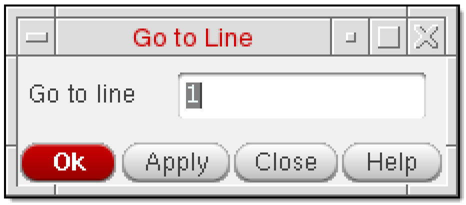
Hint : Use 'Ctrl+F' on the text editor window to find next occurence.

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

Reference: "Using Other Editing Features" on page 41

Go to Line Form

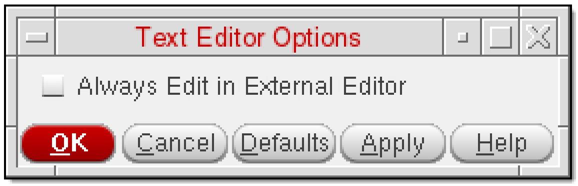
The Go to Line form lets you go to a specific line number in the displayed file.



Reference: “Showing Line Numbers and Going to a Line Number” on page 37

Text Editor Options Form

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



Form Component	Description
Always Edit in External Editor	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 39
- “useExternalEditor” on page 67

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: *Using Bookmarks and Views* in *Virtuoso Design Environment User Guide*

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: *Working with Workspaces* in *Virtuoso Design Environment User Guide*

D

Virtuoso HDL Package Setup Form

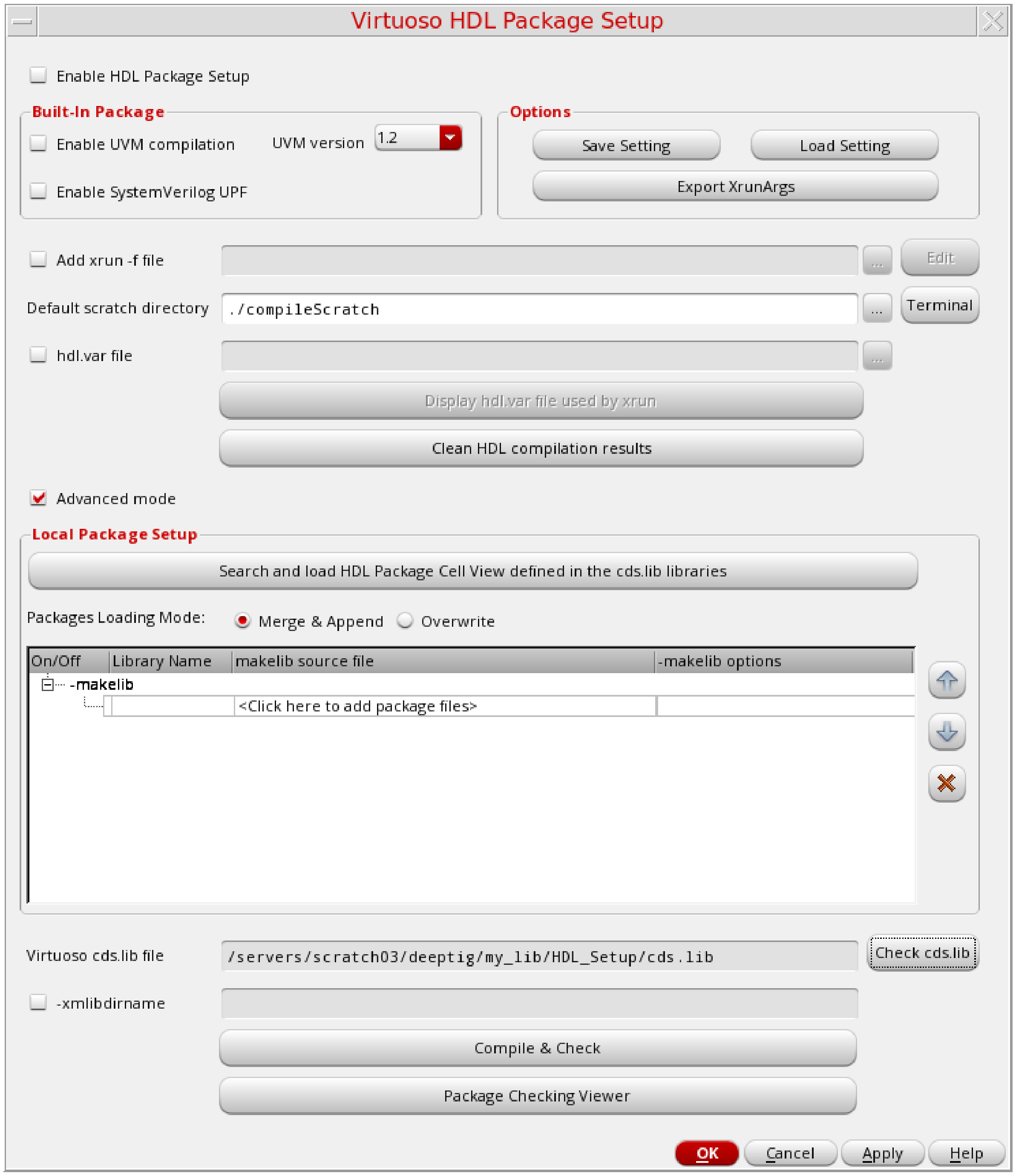
This appendix describes the Virtuoso HDL Package Setup form. The package files are important in SystemVerilog, SystemVerilog-AMS, VHDL, and VHDL-AMS for sharing and reusing common code definitions, such as functions or custom data types, between models. The form ensures that handling HDL package files 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.

Accessing the Setup Form

Click *Tools – AMS – HDL Package Setup* to open the Virtuoso HDL Package Setup form. You need to select the *Enable HDL Package Setup* check box to implement the HDL package settings defined in this form.



Virtuoso HDL Package Setup

- [ ] Enable HDL Package Setup

Built-In Package

- [ ] Enable UVM compilation — UVM version: 1.2
- [ ] Enable SystemVerilog UPF

Options

Save Setting | Load Setting | Export XrunArgs

- [ ] Add xrun -f file — Edit
- Default scratch directory: `./compileScratch` — Terminal
- [ ] hdl.var file

Display hdl.var file used by xrun | Clean HDL compilation results

- [✓] Advanced mode

Local Package Setup

Search and load HDL Package Cell View defined in the cds.lib libraries

Packages Loading Mode: [✓ Merge & Append] Overwrite

On/Off	Library Name	makelib source file	-makelib options
-makelib			
		<Click here to add package files>	

Virtuoso cds.lib file: `/servers/scratch03/deeptig/my_lib/HDL_Setup/cds.lib` — Check cds.lib

- [ ] -xmlibdirname

Compile & Check | Package Checking Viewer

OK | Cancel | Apply | Help

Ensure to enable the required options for the tools mentioned below:

- ADE
  - *Enable the HDL Package Setup.*

- *Enable the Reuse HDL package setup* option on the *Main* tab of the AMS Option form.

- VSV Netlister
  - *Enable the HDL Package Setup*.
  - *Enable the Reuse HDL package setup* option in the SystemVerilog Netlister Options form.

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

Tools	**Use of HDL Package Settings**
Text Editor	Always used.
Hierarchy Editor	Automatically used through text view check and saved when creating the `pc.db` information.
ADE/UNL	When enabled through the *Reuse HDL package setup* option on the Main tab of the AMS Option form, the package setup information is added to the regular netlist and xrun simulation options.
ADE/Simulation	Through `maestro` views. The status of the *Reuse HDL package setup* option on the Main tab of the AMS Option form is stored in the maestro view.
runams	Through `maestro` views. The status of the *Reuse HDL package setup* option on the Main tab of the AMS Option form is stored in the maestro view.
Virtuoso SystemVerilog Netlister	Always used if you enable the *Reuse HDL package setup* option in the SystemVerilog Netlister Options form.
cdsTextTo5x	Only the xrunArgs file exported from the Virtuoso HDL Package Setup form is added to `cdsHDL_XrunArgsPKG.f`.
xrun command line	Only the xrunArgs file exported from the Virtuoso HDL Package Setup form is added to `cdsHDL_XrunArgsPKG.f`.

Virtuoso HDL Package Setup Form

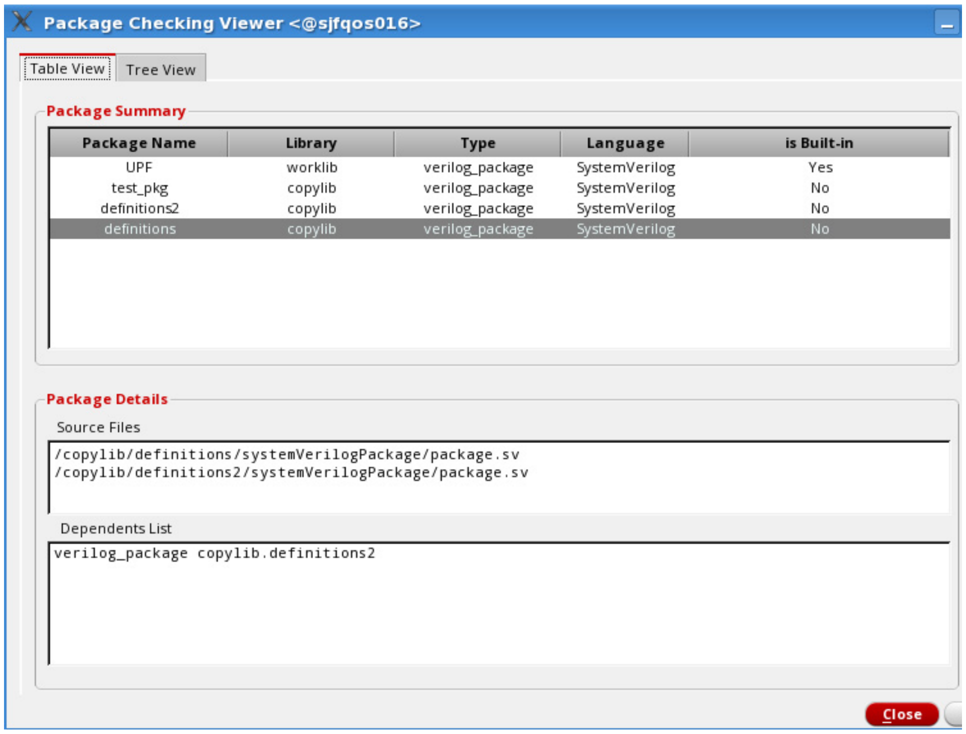
This topics describes the details of the form fields.

Field Name	Description
Enable HDL Package Setup	When selected, 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.
Built-In Package	
Enable UVM compilation	When selected, adds `-uvm -uvmhome CDNS-1.2` as one of the `xrun` compilation options.
UVM version	Specifies the UVM version from the drop-down list.
Enable SystemVerilog UPF	When selected, adds ``-makelib worklib -sv `xmroot`/tools/inca/files/1801/upf_package.sv`` as one of the `xrun` compilation options.
Options	
Save Setting	Saves the Virtuoso HDL Package Setup form settings to a specified file.
Load Setting	Loads settings from the specified file. **Note:** The file specified must be an unmodifiable file saved using *Save Setting*.
Export XrunArgs	Exports the xrunArgs file based on the Virtuoso HDL Package Setup form settings.
Add xrun -f file	Specifies the xrunArgs file, which contains user-defined xrun compilation options.
Edit	After the xrunArgs file is specified, the *Edit* button is enabled. You can click it to open and edit the file.
Default scratch directory	Specifies a directory to generate the `xcelium.d` and `xcelium` compilation results. **Note:** It is a mandatory field, and the default value is `./compileScratch`. If the directory specified does not exist, the tool creates the directory automatically before compiling.
Terminal	Launch xterm with the LINUX path specified in the *Default scratch directory* field.

Field Name	**Description**
hdl.var file	Specifies an `hdl.var` file. Recommended only when there is a project hdl.var legacy.
Display hdl.var file used by xrun	Displays the `hdl.var` file.
Clean HDL compilation results	Removes the Xcelium compilation results by running command xrun `-clean -xmlibdirpath ./compileScratch`.
Advanced mode	When selected, specifies additional package text files and `-xmlibdirname` for xrun compilation.
Local Package Setup	
Search and load HDL Package Cell View defined in the cds.lib libraries	Searches all libraries defined in `cds.lib` for package cellviews that have the following view types: ■ `systemVerilogPackageText` ■ `vhdl` and `VHDLAMSText` and view name as `package` or `body`
Packages Loading Mode	Defines the package files loading mode after the search is completed.
Merge & Append	Merges results after comparing with the existing makelib list and appends only the additional ones to the list.
Overwrite	Overwrites the existing makelib list. This option is useful when starting with a new setup or resetting entries.
makelib Table	
On/Off	Enables or disables the use of the HDL package definition in this row.
Library Name	Specifies *Library Name*.
makelib source file	Specifies package source files in this table. You can add the *makelib source file* by clicking `<Click here to add package files>`.
-makelib options	Specifies *-makelib options*.
Virtuoso cds.lib file	Specifies the `cds.lib` file.
Check cds.lib	Displays the `cds.lib` file.

Field Name	**Description**
-xmlibdirname	Specifies the name of the `xcelium.d` directory. **Note:** The `-xmlibdirname` will be used together with `-xmlibdirpath`. Therefore, the directory name specified with `-xmlibdirname` must not contain any path information.
Compile & Check	Compiles package files based on the setup specified in the Virtuoso HDL Package Setup form, including xrun -f file and hdl.var file, if specified. The compilation results and xrun.log are generated in *Default scratch directory*. 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 xrun.log file pop-up opens. You can choose to keep the compilation results or remove them by clicking *Clean HDL compilation results*.

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



Package Name	Library	Type	Language	is Built-in
UPF	worklib	verilog_package	SystemVerilog	Yes
test_pkg	copylib	verilog_package	SystemVerilog	No
definitions2	copylib	verilog_package	SystemVerilog	No
definitions	copylib	verilog_package	SystemVerilog	No

```
/copylib/definitions/systemVerilogPackage/package.sv
/copylib/definitions2/systemVerilogPackage/package.sv
```

```
verilog_package copylib.definitions2
```

- Table view: Lists all the package files compiled and found in *Default 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.