

Virtuoso Unified Custom Constraints User Guide

**Product Version IC23.1
November 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. All other trademarks are the property of their respective holders.

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

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

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information. Cadence is committed to using respectful language in our code and communications. We are also active in the removal and/or replacement of inappropriate language from existing content. This product documentation may however contain material that is no longer considered appropriate but still reflects long-standing industry terminology. Such content will be addressed at a time when the related software can be updated without end-user impact.

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

Contents

1

<u>The Constraint Manager Assistant</u>	9
<u>Constraints Workspace Configuration</u>	10
<u>Constraints-Helper Workspace Configuration</u>	10
<u>An Introduction to Constraints</u>	11
<u>Virtuoso Unified Custom Constraints - A Use Model Overview</u>	12
<u>Constraint Manager Overview</u>	12
<u>Circuit Prospector Overview</u>	13
<u>The Constraint View</u>	14
<u>Constraint Name Column</u>	15
<u>Changing a Constraint View Name List</u>	16
<u>Constraint Storage</u>	17
<u>Ensuring that the Constraint View Is Editable</u>	17
<u>Introducing the Constraint Manager Assistant</u>	18
<u>Accessing the Constraint Manager</u>	21
<u>Constraint Manager Toolbar</u>	22
<u>Customizing Constraint Manager Assistant Content</u>	47
<u>Module Generator and Process Rule Editor Constraint Manager Options</u>	48
<u>Updating Constraints</u>	49
<u>Editing a Constraint in Associated Constraint Generator Dialog Box</u>	51
<u>Constraint Browser</u>	52
<u>Constraint Filters</u>	53
<u>Constraint Browser Features</u>	61
<u>Constraint Browser Context-Menus</u>	65
<u>Constraint Manager Options Form (Schematic)</u>	72
<u>Constraint Manager Options Form (Layout)</u>	76
<u>Constraint Browser Columns</u>	81
<u>Creating Constraints in the Constraint Manager</u>	89
<u>Constraint Creation Basics</u>	89
<u>Steps to Create Constraints in the Constraint Manager</u>	91
<u>Creating Constraints Using the BitSelector</u>	93

Virtuoso Unified Custom Constraints User Guide

<u>Creating Constraints for Iterated Objects</u>	93
<u>Creating Modgen Constraints</u>	95
<u>Creating Constraints at Different Design Levels</u>	128
<u>Viewing Constraints in the Constraint Manager</u>	131
<u>Moving Constraints to Current or Sub-Cells</u>	131
<u>Overriding Constraints</u>	138
<u>Constraint Creation History</u>	140
<u>Virtuoso Custom Digital Placer and Creating Constraints on Pins/Pin Figures</u>	142
<u>Constraint Templates</u>	143
<u>Deleting Constraints</u>	156
<u>Saving and Storing Constraints (Schematics Only)</u>	157
<u>Switching On/Off Constraint Menu Options in Schematics XL File Menu</u>	157
<u>Selection and Cross-Selection of Constraints</u>	158
<u>Halo Highlighting</u>	158
<u>Configuring Constraint Halo Display</u>	161
<u>Adding Constraint Members</u>	170
<u>Deleting Constraint Members</u>	171
<u>Interactive Constraint Searching</u>	172
<u>Constraint Parameter Editor</u>	173
<u>Selecting and Displaying Constraints and Constraint Members in the Constraint</u>	
<u>Parameter Editor</u>	174
<u>Constraint History</u>	175
<u>Editing Constraint Parameters</u>	176
<u>Constraint Parameter Editor Status Visualization</u>	182
<u>Transferring Constraints Between Schematics and Layout</u>	184
<u>Using Schematic Constraints in Layout</u>	186
<u>Pushing and Pulling Constraints in a Schematic or Layout Hierarchy</u>	187
<u>Constraints vs. Rules</u>	191
<u>Using Constraint Notes</u>	194
<u>System-Defined Constraint Notes</u>	194
<u>User-Defined Constraint Notes for Constraints</u>	215
<u>User-Defined Constraint Notes for Templates</u>	217
<u>Registering Pre- and Post-Create Triggers for Constraint Notes</u>	220
<u>Creating Constraint Notes for a Complete Hierarchy</u>	221
<u>Working with Constraint Views in a Design-Managed Environment</u>	222
<u>Constraint Manager in Layout XL</u>	230

<u>Updating Constraints From Schematic to Layout</u>	232
<u>Comparing Constraints between Schematic and Layout</u>	232
<u>Overriding Process Rules</u>	239
<u>Creating a Cluster Boundary Using the Constraint Manager</u>	239
<u>Creating and Editing Net Constraints</u>	242
<u>Making Net Connections</u>	247

2

The Circuit Prospector Assistant

<u>An Introduction to the Circuit Prospector Assistant</u>	249
<u>Motivation</u>	250
<u>Storage</u>	251
<u>Accessing the Circuit Prospector</u>	251
<u>Circuit Prospector User Interface</u>	251
<u>The Circuit Prospector Constraint Creation Flow</u>	263
<u>Step 1: Select a circuit prospector category</u>	263
<u>Step 2: Select a finder to search for relevant design objects</u>	264
<u>Step 3: (Optionally) Filter/ select the found objects to apply constraints to</u>	265
<u>Step 4: Apply default action/ constraints to the selected objects</u>	269
<u>Circuit Prospector Flow Summary</u>	271
<u>Edit Finder Form</u>	272
<u>Edit Iterator Form</u>	274
<u>Edit Generator or Action Form</u>	276
<u>Capture Circuit Structure Form</u>	278
<u>Capturing New Structures</u>	281
<u>Editing Structures</u>	283
<u>Deleting Structures</u>	284
<u>Using Repeatable on a Captured Structure</u>	285
<u>Edit Match Expression Form</u>	287
<u>Edit Terminal Name Expression Form</u>	288
<u>Customization of the Circuit Prospector</u>	294
<u>Categories</u>	300
<u>Creating and Editing Categories</u>	363
<u>Detecting Current Paths and Creating Constraints</u>	370
<u>Finders</u>	373

Virtuoso Unified Custom Constraints User Guide

<u>Creating Finders</u>	374
<u>Editing Finders</u>	377
<u>Deleting Finders</u>	378
<u>Iterators</u>	379
<u>Pre-Defined Iterators</u>	380
<u>Creating Iterators</u>	383
<u>Editing Iterators</u>	385
<u>Deleting Iterators</u>	386
<u>Constraint Generators</u>	387
<u>Pre-Defined Constraint Generators</u>	388
<u>Creating Constraint Generators</u>	396
<u>Editing Constraint Generators</u>	398
<u>Deleting Constraint Generators</u>	399
3	
<u>The Process Rule Editor</u>	401
<u>Starting the Process Rule Editor</u>	403
<u>Process Rule Editor Graphical User Interface</u>	417
<u>Toolbar</u>	418
<u>Scope Selection Section</u>	419
<u>Browser Pane</u>	419
<u>Create Constraint Group Section</u>	423
<u>Create Process Rule Section</u>	424
<u>Dynamic Widget</u>	425
<u>Apply Constraint Group Section</u>	429
<u>Creating a Constraint Group</u>	430
<u>Customizing a Constraint Group</u>	434
<u>Creating a New Process Rule</u>	435
<u>Creating a New Process Rule in the Technology or Design Mode</u>	436
<u>Creating a New Process Rule for a Selected Object</u>	437
<u>Creating a New Process Rule for a Constraint</u>	441
<u>Copying an Existing Process Rule</u>	445
<u>Creating a Layer-Purpose Pair</u>	446
<u>Applying Valid LPPs Process Rule</u>	449
<u>Creating a Voltage Dependent Process Rule</u>	450

Virtuoso Unified Custom Constraints User Guide

<u>Editing Existing Process Rule Values</u>	452
<u>Applying a Constraint Group</u>	455
<u>Setting the Precedence of a Constraint Group or Process Rule</u>	459
<u>Virtuoso Generated Constraint Groups</u>	460
<u>Applying a Virtuoso Generated Constraint Group</u>	465
<u>Applying an Existing Constraint Group to a Selected Object</u>	469
<u>Removing a Constraint Group or Process Rule</u>	474
<u>Saving Changes and Closing the Process Rule Editor</u>	478

A

<u>Default Constraint Types</u>	479
<u>Constraints in Virtuoso Schematic and Layout Editing Environment</u>	480
<u>Alignment</u>	483
<u>Area Pin Group Guide</u>	489
<u>Bus</u>	492
<u>Cell Boundary</u>	494
<u>Cluster</u>	499
<u>Cluster Boundary</u>	503
<u>Correlation</u>	505
<u>Diff Pair</u>	506
<u>Disable Permutation</u>	509
<u>Distance</u>	510
<u>Edge Pin Group Guide</u>	516
<u>Fixed</u>	519
<u>Guard Ring</u>	520
<u>High Precision C Extraction</u>	526
<u>High Precision R Extraction</u>	527
<u>Locked</u>	528
<u>Matched Capacitance</u>	530
<u>Matched Instance Color</u>	531
<u>Matched LDE Parameters</u>	532
<u>Matched Length</u>	537
<u>Matched Orientation</u>	540
<u>Matched Parameters</u>	547
<u>Max Capacitance</u>	553

Virtuoso Unified Custom Constraints User Guide

<u>Max Coupling Capacitance</u>	555
<u>Max Resistance</u>	556
<u>Max Voltage Drop</u>	558
<u>Modgen</u>	562
<u>Net Class</u>	566
<u>Net Class Group</u>	573
<u>Net Class Hier Group</u>	577
<u>Net Priority</u>	588
<u>Orientation</u>	590
<u>Placement Path</u>	597
<u>Process Rule Overrides</u>	600
<u>Rail</u>	606
<u>Shielding</u>	609
<u>Symmetry</u>	618
<u>Voltage Synced Net</u>	624
<u>Delta Voltage</u>	626
<u>Assigning a Wire Type to a Net</u>	627
<u>Specifying the Color Mask for Critical Nets in Schematic</u>	631
<u>Using colorMask Constraint Parameter of msSameMask</u>	632
<u>Constraints in Virtuoso Parasitic Aware Design Environment</u>	633
<u>Parasitic Estimate</u>	633
<u>Parasitic Filter</u>	635
<u>Axis Parameter</u>	638
<u>Widths Minimum Parameter</u>	640
<u>Overriding the Widths Parameter Values</u>	640
<u>Allowed Width Ranges Form</u>	640
<u>Spacing Parameter</u>	643
<u>Overriding the Spacing Parameter Values</u>	644
<u>Parameter Types</u>	653
<u>Support of Shapes</u>	653

B

<u>Troubleshooting Constraints</u>	655
<u>Performance Issues</u>	656
<u>Read-Only Issues</u>	657

Virtuoso Unified Custom Constraints User Guide

Transfer and Migration Issues	658
Constraint Conflict Issues	660

C

[Environment Variables](#) 663

[Setting Environment Variables](#) 663

[.cdsenv File](#) 663

[.cdsinit File](#) 664

[CIW](#) 664

[Displaying the Current Value of an Environment Variable](#) 664

[List of Environment Variables](#) 665

[addParallelDevices](#) 667

[autoConstraintNotes](#) 668

[autoSelectTemplateConstraint](#) 669

[comanageConstraintView](#) 670

[comanageFollowModeOnDescend](#) 671

[comanageFollowModeOnOpen](#) 672

[consistencyFailureDefaultWarning](#) 673

[constraintMenuHistorySize](#) 674

[constraintNotesSyncOnDesignCheck](#) 675

[constraintNotesSyncOnDesignOpen](#) 676

[crossCoupleDiffPairRequireSourceNotSupply](#) 677

[crossHalo](#) 680

[currentMirrorIncludeDummies](#) 681

[defaultViewName](#) 682

[diffPairRequireSourceNotSupply](#) 683

[disableCICustomCGPopup](#) 685

[distanceMFactorExpansion](#) 686

[editTransferredTemplates](#) 688

[highlightCPResults](#) 689

[keepFigGroupMembersWithinSameCluster](#) 690

[matchedToIgnore](#) 691

[maxMatchesPerDevice](#) 692

[noteTooltipMaxCharWidth](#) 693

[noteTooltipMaxLines](#) 694

Virtuoso Unified Custom Constraints User Guide

<u>purgeOnCloseCellView</u>	695
<u>restrictNetGroupMembership</u>	696
<u>updateConstraints</u>	697
<u>useOldComparisonReport</u>	699
<u>viewNameList</u>	701
<u>viewWithConfigCell</u>	702

The Constraint Manager Assistant

The Cadence® Virtuoso® unified custom constraint management system allows you to establish design needs, save them as constraints, and share those constraints across specification, simulation, and implementation to drive the accelerated layout solution with reduced errors. A constraint-driven design preserves the design intent by enabling efficient design collaboration.

The unified custom constraint management system is available in:

- Virtuoso® Schematic Editor XL
- Virtuoso® Layout Suite XL and higher tiers

Note: Other applications, such as Virtuoso® ADE, also use constraints.

Constraint location and creation functionality is available in the above applications using [The Circuit Prospector Assistant](#) and [The Constraint Manager Assistant](#).

The *Constraint Manager* and the *Circuit Prospector* assistants can be accessed from the [Constraints Workspace Configuration](#) and the [Constraints-Helper Workspace Configuration](#) as well as through the menu and toolbar selections discussed later (see [Accessing the Constraint Manager](#) and [Accessing the Circuit Prospector](#)).

 Important

This user guide focuses on the creation and use of constraints in the *Constraint Manager* and *Circuit Prospector* assistants in Schematics XL. For more specific information on physical design constraints and their uses in Layout XL, see [Constraint Manager in Layout XL](#).

This user guide is aimed at developers and designers of integrated circuits and assumes that you are familiar with:

- The Virtuoso Studio Design Environment and application infrastructure mechanisms designed to support consistent operations between all Cadence® tools.
- The applications used to design and develop integrated circuits in the Virtuoso Studio Design Environment, notably, the Virtuoso Layout Suite, and Virtuoso Schematic Editor.

- The Virtuoso Studio Design Environment technology file.
- Component description format (CDF), which lets you create and describe your own components for use with Layout XL.

Constraints Workspace Configuration

The *Constraints* workspace configuration assists you in the creation, selection, and editing of constraints.

You can access the *Constraints* workspace from the Virtuoso® Layout Suite XL, and the Virtuoso® Schematic Editor XL by using one of the following methods:

- Selecting *Constraints* from the drop-down list box in the *Workspace Configuration* toolbar.
- Selecting *Window – Workspaces – Constraints*.

The *Constraints* workspace comprises of the following assistants:

- *The Constraint Manager Assistant*
- *The Navigator assistant*

For more information, see the *Navigator Assistant* in the *Virtuoso Schematic Editor User Guide*

- *The Property Editor assistant*

For more information, see the *Property Editor* in the *Virtuoso Schematic Editor User Guide*

Note: Additionally, in VLS XL, the *Layers* and *Objects* assistants are also displayed.

For more information on workspaces, see the *Getting Started with Workspaces* and *Working with Workspaces* chapters in the *Virtuoso Studio Design Environment User Guide*.

Constraints-Helper Workspace Configuration

The *Constraints-Helper* workspace configuration assists you in the creation of constraints.

You can access the *Constraints-Helper* workspace from the Virtuoso® Schematic Editor XL (only) by using one of the following methods:

- Selecting *Constraints-Helper* from the drop-down list box in the *Workspace Configuration* toolbar.
- Selecting *Window – Workspaces – Constraints-Helper*.

The *Constraints-Helper* workspace comprises of the following assistants:

- *The Constraint Manager Assistant*
- *The Circuit Prospector Assistant*

For more information on workspaces, see the *Getting Started with Workspaces* and *Working with Workspaces* chapters in the *Virtuoso Studio Design Environment User Guide*.

An Introduction to Constraints

A simple way to define a constraint is to consider it as an objective that can be used to influence the possible implementation, placement, or routing in a system. In the context of layouts and schematics, this would be a design object with respect to its goal. Constraints are therefore requirements that need to be followed during design implementation.

This document refers to designs as being the full set of design objects in a design hierarchy, including all instances, nets, terminals, pins and instance terminals expressed as cell, instance or occurrence. By understanding constraints in your design hierarchy, you can determine specification limits for design objects.

A constraint can have parameters that are pre-defined name-value pairs that further define the design intent and the status of that constraint.

Note: Constraints are stored in the layout database or in *The Constraint View* (not the schematic view) for schematics.

Virtuoso Unified Custom Constraints - A Use Model Overview

The Virtuoso Unified Custom Constraints system provides for constraints to be captured alongside connectivity in schematic applications to form a full design specification. This unique capability allows electrical designers to capture additional concerns to ensure that the electrical performance of their design is guarded against unwanted electrical side-effects during implementation.

These additional physical requirements can include:

- Degrees of device matching
- Nets that require symmetrical routing
- Nets that require shielding
- Capacity to carry current, guard rings and so on.

This formal specification facilitates a vastly improved dialog between front-end and back-end engineers.

Constraint Manager Overview

To provide for this improved dialog, the *Constraint Manager* is available as a docked assistant inside both the *Virtuoso Schematic Editor* (VSE) and the *Virtuoso Layout Suite* (VLS) at the XL tier and above, displaying specification and implementation constraints respectively.

The history of specification and implementation constraints are tracked via *creation*, *last modified*, and *last checked* timestamps (see [Constraint History](#)) that also identify the reason for the change and the person or application that made that change. Additional implementation constraints can be directly entered into the VLS *Constraint Manager* by the back-end engineer.

Constraints can also be overridden (see [Overriding Constraints](#)), and this state is clearly marked so that the specification and implementation can be brought back into alignment with further discussion between front-end and back-end engineers.

Devices can also be carefully specified in the VSE *Constraint Manager* via direct access to the VLS modgen device builder. Specifications can also be applied to cells as well as designs (see [Creating Constraints at Different Design Levels](#)), providing stronger control of IP as well as designs.

In summary, the *Constraint Manager* enables the specification to be carefully managed and implementation progress be tracked.

Constraints Database

The provision of a unified constraints database, accessible using the *Constraint Manager*, provides constraint storage for and access to, for example, many types of constraints including:

- sizing and optimization constraints
- floorplanning constraints
- interactive and auto-placement constraints
- interactive and auto-routing constraints

Circuit Prospector Overview

Virtuoso Unified Custom Constraints offers a valuable assistant in VSE XL to automate or assist with the constraint create process for designers. The Circuit Prospector Assistant is capable of finding circuit sub-structures in a design then generating the right set of constraints. Structures covered as standard include cascades, current mirrors, differential pairs, and so on.

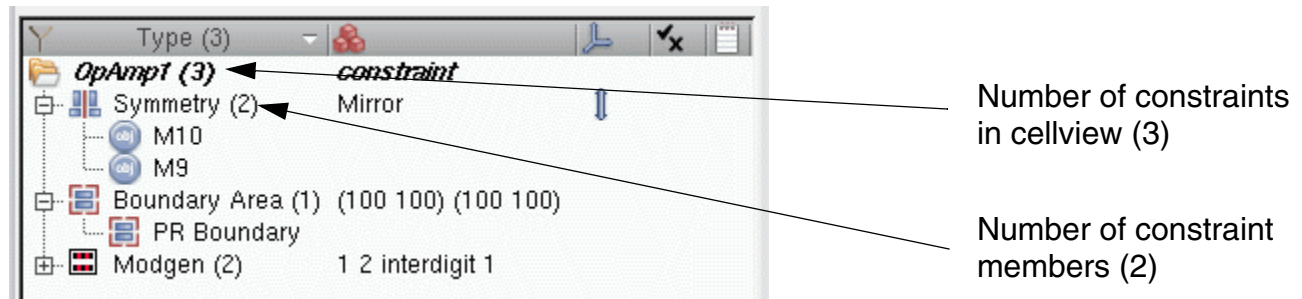
As the *Circuit Prospector* is extensible, many other structures can be added to *Circuit Prospector's* library by CAD departments and designers.

If you have a structure in an existing design (and have already populated it with a set of appropriate constraints), the *Circuit Prospector* can snapshot the structure and its constraints, and add it to its library. The *Circuit Prospector* provides the means to generate constraints quickly, accurately, and repeatedly. The barriers to constraint driven design have therefore been significantly reduced with the introduction of the *Circuit Prospector*.

The various components of the constraint-driven flow, from constraint specification to implementation by applications and automatic status tracking, help to accelerate the design process significantly and augment specification-driven design.

The Constraint View

The constraint view name (default *constraint*) is displayed in the *Parameters* column in the *Constraint Manager*.



Note:

- If the constraint view is in read-only or edit mode and unedited, the constraint names and related information are displayed in black. The name of the constraint view appears in **bold**.
- If the constraint view has been edited and not saved yet, its name appears as ***bold italic***.

See also: [Creating Constraints in the Constraint Manager](#).

Multiple constraint views can be associated with each schematic. In terms of *Save*, *Save a Copy*, *Discard Edits*, and *Close*, constraint views remain in sync with the associated schematic.

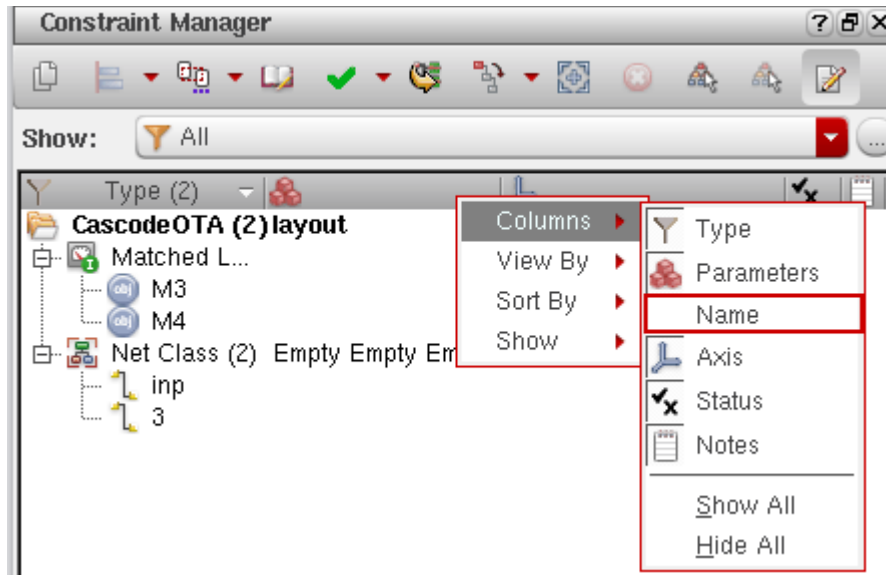
Note: You can associate a constraint view with a single schematic view of the same cell by setting the `CI_SAME_CELL_UNIQUE_SCHEMATIC_CACHE` environment variable.

A difference can arise when the design hierarchy is read-only or unmodified, but the constraint view has been modified. Allowing constraint changes without modifying the design is an important reason for storing constraints separately from the database cellviews.

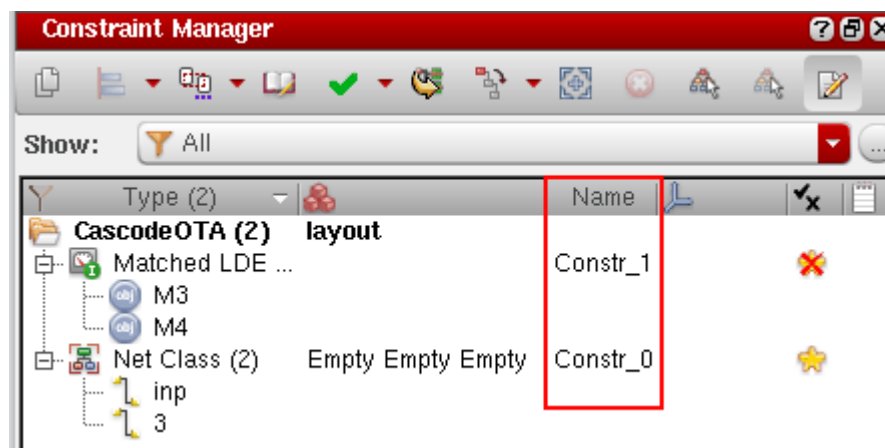
Note: The constraints view for the current cell is opened in *append* (write) mode, but the constraints associated with the sub-cells are opened in read-only mode.

Constraint Name Column

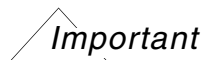
Constraint names can be displayed as a column in the Constraint Manager. To view the shortcut menu, right-click the filter and select *Column — Name* option, as shown below:



The constraint names are displayed in the new *Name* column of the Constraint Manager assistant.



Changing a Constraint View Name List



The default constraint view name used is “`constraint`”. The rest of this section will be of interest only if you need more than one constraint view per schematic or if you require to use alternative names for your constraint views.

You can specify the constraint views that you want to use in your design by modifying the global constraint view list, where by default “`constraint`” is used as the view name (see [Constraint Manager Options Form \(Schematic\)](#) for information on the *Global Constraint View List* option).

However, when generating a layout or running a simulation, the constraint view to be used needs to be specified in the constraint view list of the physConfig (through CPH, see [Configure the Physical Hierarchy](#) in the Virtuoso Layout Suite documentation) and the config (through HED, see [Cadence Hierarchy Editor User Guide](#)). If nothing is specified, the same view names are used in both cases.

When VLS XL (and higher tiers) is launched, VSE XL picks up the constraint view list that has been set in the physConfig. Sub-configurations can also be created.

Note: You can also configure constraint views using the Cadence Hierarchy Editor (see the [Cadence Hierarchy Editor User Guide](#)).

On the rare occasion that a config cannot be used, or does not exist, constraint views can also be configured globally using the `viewNameList` environment variable.

Note: It is still recommended that you use “`constraint`” as the default constraint view name, and perhaps create additional constraint view names when overrides are required.

Constraint Storage

Constraint storage differs from schematic to layout. In the schematic, constraints are stored in an OpenAccess database (`hierDesign.oa`) that resides in a view called constraint view; whereas, in the layout, the constraints are stored in the layout database itself.

When you open in Schematic XL a design that does not have associated constraints, a constraint view is not created. A constraint view is created when you:

- Create the first constraint by using:
 - ❑ Constraint Manager assistant
 - ❑ Circuit Prospector assistant
- Run the following SKILL functions:
 - ❑ `ciConCreate`
 - ❑ `ciConCreateExpanded`
- Transfer constraints from layout to schematic.
- Load constraints from another view.
- Create an axis by using `ciAxisCreate`.
- Open Process Rule Editor.
- Create a process rule.
- Create a design Constraint Group.

Ensuring that the Constraint View Is Editable

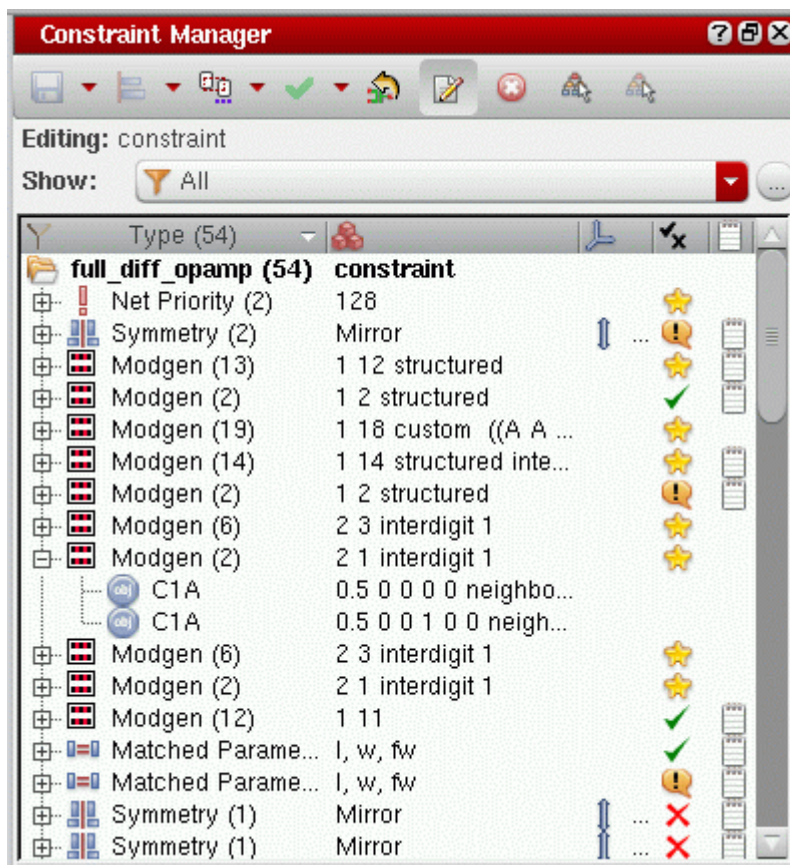
Checking out a schematic view does not check out the (companion) constraint view automatically and the constraints remain grayed out in the Constraint Manager assistant. Therefore, if you want to open the constraint view in edit mode, you must select the *Make Editable* option from the *Save Constraints* drop-down list box on the *Constraint Manager* toolbar.

Note: If the schematic view is initially opened in read-only mode, the constraints displayed in the Constraint Manager assistant appear inactive.

Introducing the Constraint Manager Assistant

The *Constraint Manager* dockable assistant pane allows for the addition, modification, and deletion of constraints from/to the Cadence Constraint Storage system. The *Constraint Manager* user interface comprises of two main component parts; Constraint Browser and Constraint Parameter Editor.

The *Constraint Manager* is available as an assistant in both schematics and layout. In Schematic XL, as shown in the figure below, the *Constraint Manager* displays the content of the constraint view being used as per a given configuration.



In Layout XL, the *Constraint Manager* displays the constraints that currently exist in the layout.

The *Constraint Manager* displays a full set of constraints for a design wherever you are in the hierarchy with the current cellview, and wherever the constraints were created in that hierarchy. The *Constraint Manager* displays a full set of constraints for the current cellview and below.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Note: If you are in the Virtuoso Layout Suite, the hierarchy displayed in the *Constraint Manager* (and the *Navigator*) will be controlled by the CPH ([Configure Physical Hierarchy](#)).

Only those cells at the current level can be edited (those with a white background). Constraints, with a gray background, that are found at a lower level cannot be edited.

Examples of constraints include matching, symmetry, orientation, matched orientation, IR drop, clustering, alignment, area, distance, boundaries, guard rings, custom module generators (modgens), cell plans, and clusters.

The *Constraint Manager* also keeps track of all added constraints by grouping constraints by instance or constraint type. Each constraint can be modified, for example symmetry can be changed from mirror to simple.

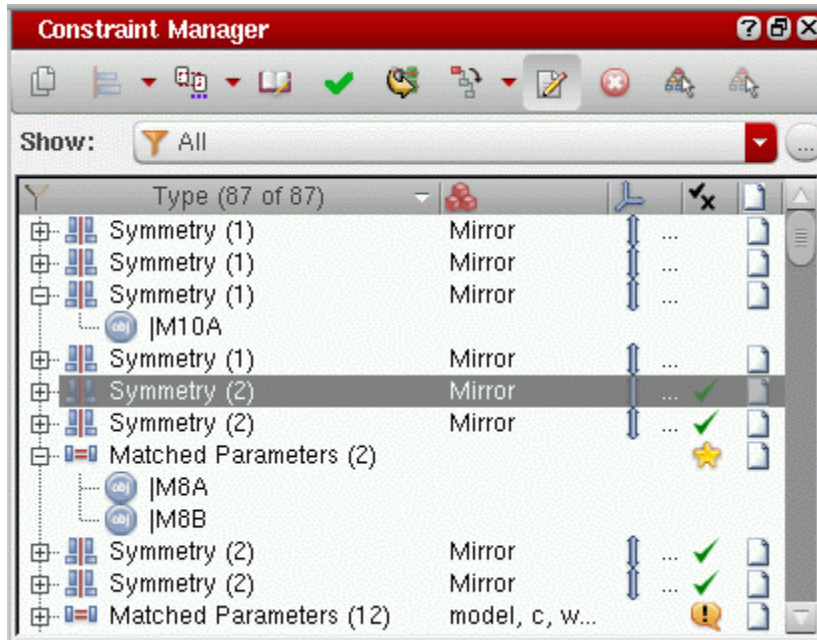
The *Constraint Manager* allows designers to manually create constraints by selecting groups of instances, nets, pins, and terminals from the design canvas and applying constraints to them. For example, a designer may look to apply a set of constraints to common circuit structures, related devices (such as the same size, area, or type), devices with shared characteristics and connectivity, and symmetrical devices with specific properties. For information on how this information is captured see [The Circuit Prospector Assistant](#).

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Important

You can only have one Constraint Manager assistant active in each session window. If you open multiple top-level cellviews within a session, the Constraint Manager assistant updates to display the design constraints for the currently active top-level cellview.



The figure above shows the Constraint Manager assistant in Layout XL. Some of the Constraint Manager Toolbar options differ from those available in Schematics XL. In addition, the constraints created using the *Constraint Manager* in Schematics XL will also be reflected (transferred) in the *Constraint Manager* in Layout XL (for more information on this, see Transferring Constraints Between Schematics and Layout in the first instance).

Accessing the Constraint Manager

You can access the *Constraint Manager* by:

- Selecting *Window – Assistants – Constraint Manager*.
- Selecting *Constraint Manager* from the *Toolbars* context-sensitive menu.

Important

The *Constraint Manager* is also automatically loaded when you select the *Constraints* or *Constraints-Helper* (schematics only) default workspaces from the drop-down list box on the *Workspace Configuration* toolbar (see also the [Constraints Workspace Configuration](#) and the [Constraints-Helper Workspace Configuration](#)).

Constraint Manager Toolbar

The Constraint Manager toolbar varies when accessed from Virtuoso Schematic XL and Virtuoso Layout XL. The images below show the difference.

Figure 1-1 Constraint Manager toolbar (Virtuoso Schematic Editor XL)

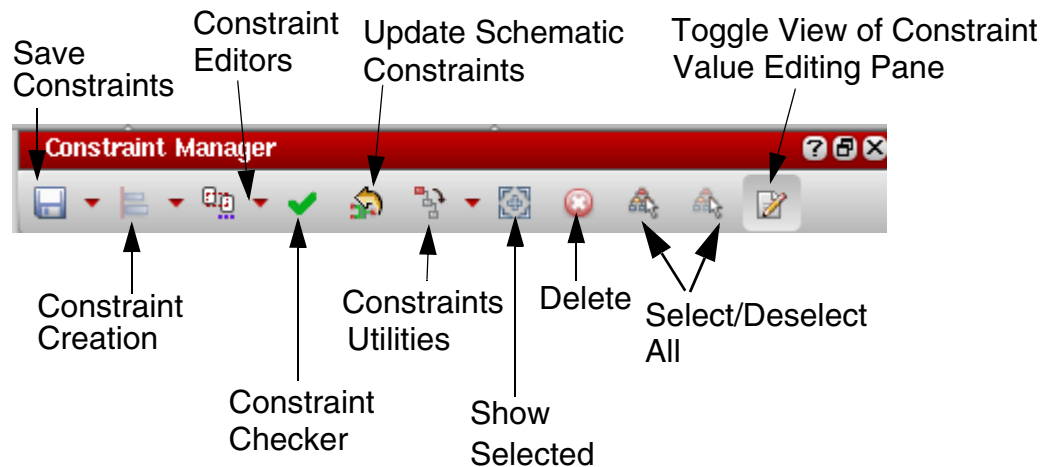
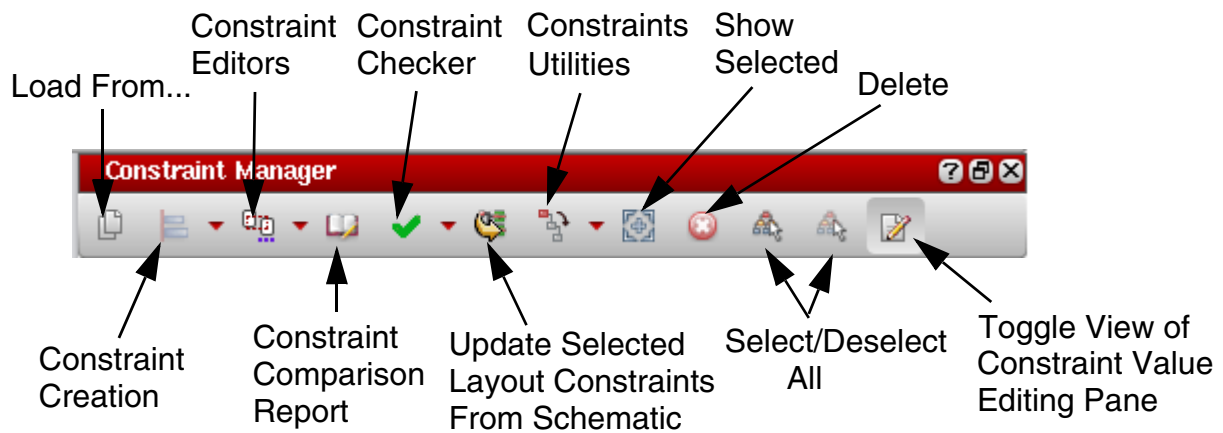


Figure 1-2 Constraint Manager toolbar (Virtuoso Layout XL)



Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

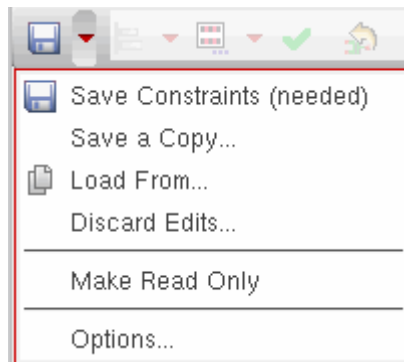
The *Constraint Manager* toolbar provides the following options (available in both schematics and layout unless otherwise stated):

Save Constraints
drop-down
(*schematics only*)

Saves the constraints listed in the *Constraint Manager* to the constraint view that is currently in use (and is configured) with the given schematic.

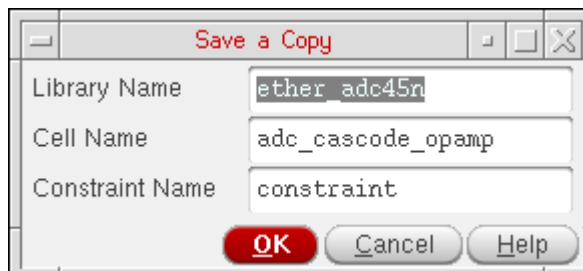
The main VSE menu *File – Save/Save a Copy/Discard Edits* options also update for constraint use. For example, *Save (Constraint)* will display when a constraint update has been made but not yet saved. Consequently, the *File* menu save options can also be used to save constraint status.

Click the *Save Constraints* drop-down list box for the following options:



- *Save a Copy*
(*schematics only*)

Invokes the Save a Copy form which allows you to save a copy of the current view. This allows you to save different constraint views (see [The Constraint View](#)).



Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

- *Load From*
(option available in
schematics and as a
standalone toolbar
option in layout)

Invokes the Load Constraints form.

The *Load From* option allows you to populate the Constraint Manager assistant with constraints of a corresponding cellview (including any saved templates), from another library.

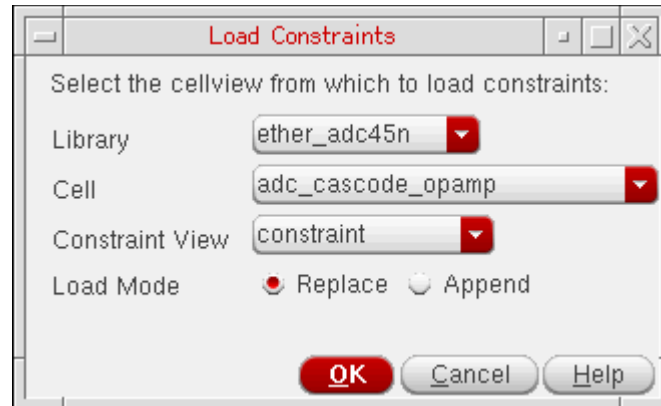


Figure 1-3 Load Constraints from an External View



Tip

Alternatively, you can use the `ciLoadConstrFrom` SKILL function to load the constraints from the source cache to the target cache of the same type.

Any constraint inconsistencies or corruptions will be cleared when constraints are loaded. In such cases, a warning message is also displayed in the CIW advising you of this action.

On completion of the constraints loading process, an INFO message is displayed in the CIW listing the count of constraints, parasitic constraints, and constraint groups that were successfully loaded from the specified cellview.

In VSE, the constraints can be loaded from a constraint cellview, while in VLS constraints can be loaded from a layout cellview. To be relevant, and in-context, the members of the loaded constraints must have the same instance or net names as those in the current cellview.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

When *Replace* mode (the default) is selected, all constraints in the Constraint Manager assistant will be purged from the cache before the new constraints are loaded.

If *Append* mode is selected, the new constraints from the selected cellview will be added to the existing constraints in the Constraint Manager assistant, except where the consistency checker detects a conflict with an existing constraint. If the constraint name conflicts with an existing constraint name, the loaded constraint will be renamed. However, if the consistency checker detects an incompatibility between an existing constraint and a new constraint, the loaded constraint will not be created.

Note: In VLS XL, the *Load From* option is available directly from an icon on the Constraint Manager assistant toolbar. Additionally, in VLS XL, the *Constraint View* field name in the Load Constraints form is labeled as *Layout View*.

- *Discard Edits* (*schematics only*)

Select to discard any edits made to constraint values since the last save.

The Discard Edits pop-up window is displayed asking you to confirm that you want to discard edits.

The *File – Discard Edits (Constraint)* option in the schematic editor can also be used to discard any constraint edits made. A discard edits form is also displayed for confirmation.

- *Make Read Only/Editable* (*schematics only*)

Select to change the constraint view's mode from edit to read-only, or conversely. When you choose to switch the constraint view to read-only mode, the Make Read Only form is displayed. This form allows you to choose whether to save or discard your changes, if any, before making the constraint view read-only.



Alternatively, you can use the *File – Make Read Only/Editable* option in the schematic editor to change edit mode. You can also use the *File – Make Read Only* menu from the CIW to display the Make Read Only form and select the constraint view on which edit permissions need to be revoked.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Checking out a schematic view does not automatically trigger a check out of the (companion) constraint view. Therefore, you might see that the constraint view remains grayed out in the *Constraint Manager* even after you have checked out the schematic view. To ensure that the constraint view is in edit mode, you must select the *Make Editable* option from the *Save Constraints* menu on the *Constraint Manager* toolbar. This assumes that the schematic view was not initially opened in read-only mode.

Note: When a constraint view is in read-only mode, for example after setting to *Make Read Only*, the *Constraint Creation* menu and the *Update Schematic Constraints* and *Delete* buttons appear disabled. The text of the constraints view directory, and the text of the constraints contained directly within, are also be grayed out.

- *Options*
(*schematics only*)

Displays the Constraint Manager Options form.

Constraint Creation
(*schematics and layout*)

Creates the selected constraint comprising of the currently selected object group members (as selected in the design canvas or the Navigator Assistant).

Constraints that cannot be generated with the current object selection will be grayed out. For detailed information, see Constraint Creation and Current Object Selection and Constraint Creation History.

Individual constraints are categorized and located in the following related constraint sub-menus: *Electrical*, *Placement*, or *Routing*. For more information on what constraints can be found in each constraint category and specific constraint creation conditions that need to be met for each constraint, see The Constraint Manager Assistant.

In addition to these three constraint sub-menus, you have the *Rapid Analog Prototype* sub-menu. The constraint generators available under this sub-menu are the same as:

- Generators associated with all finders available in the Rapid Analog Prototype category of the Circuit Prospector assistant.

- Generators associated with the *Series Resistor Array*, *Parallel Net Resistor Array*, *Parallel Resistor Array (Length)*, and *Block Resistor Array* finders available in the *Structures* category of the Circuit Prospector assistant. See also the *Creating Modgen Template for Resistor Arrays* section for procedural details.



Tip

The *Parallel Net Resistor Array* and *Parallel Resistor Array (Length)* finders have the same generator, that is, *ParallelResistorArray*.

Note: When the Constraint Manager assistant is opened in read-only mode, the *Constraint Creation* menu will be grayed out to prevent creation of any constraints. For more information on how to create constraints, see *Creating Constraints in the Constraint Manager*.

See *Customizing Constraint Types Using a Configuration File* in the *Virtuoso Unified Custom Constraints Configuration Guide* for information on:

- controlling the constraint types and parameter names that should be displayed.
 - how to display constraints in a hierarchical menu format in the *Constraint Creation* menu.
- *Add Design Intent* Lets you create a constraint on a design intent using the *Constraints Constraint Manager*. For more information on design intents, refer to the *Virtuoso Design Intent User Guide*.

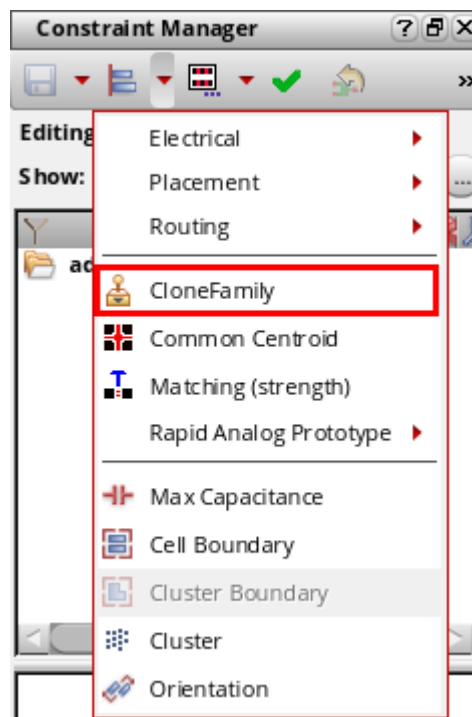
Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

- *CloneFamily*

A *Clone Family* template can be used to create cloning constraint on the selected devices. A clone is an identical set of devices with the same topology. For detailed information on cloning and clone families, refer to the Generating Clones section of the *Generating a Layout* chapter in the Virtuoso Layout Suite documentation.

Select some devices on the canvas and the *CloneFamily* submenu gets enabled in the *Constraint Creation* menu as shown below.

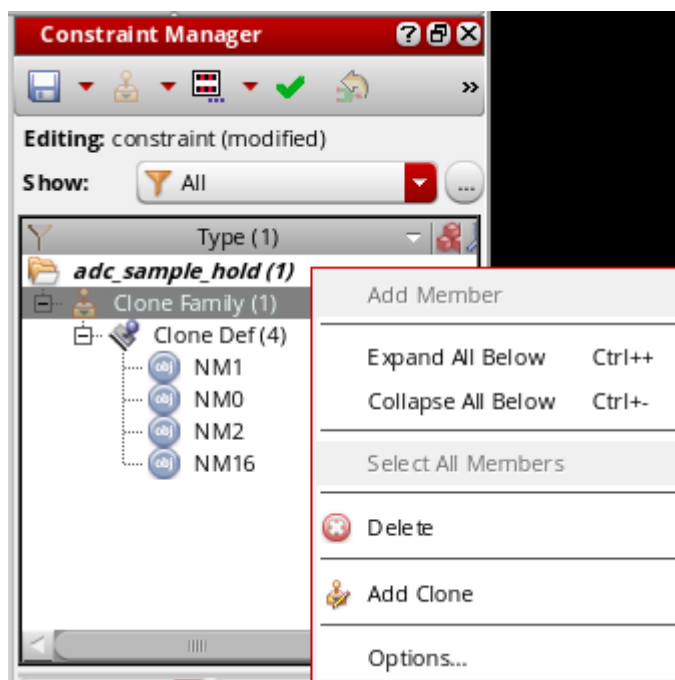


The *Clone Family* template is created with a *Clone Def* constraint on the selected devices.

Adding Clone Def to an Existing Clone Family Template

To create one or more clones in the schematic with the same topology as the selected clone source, do the following:

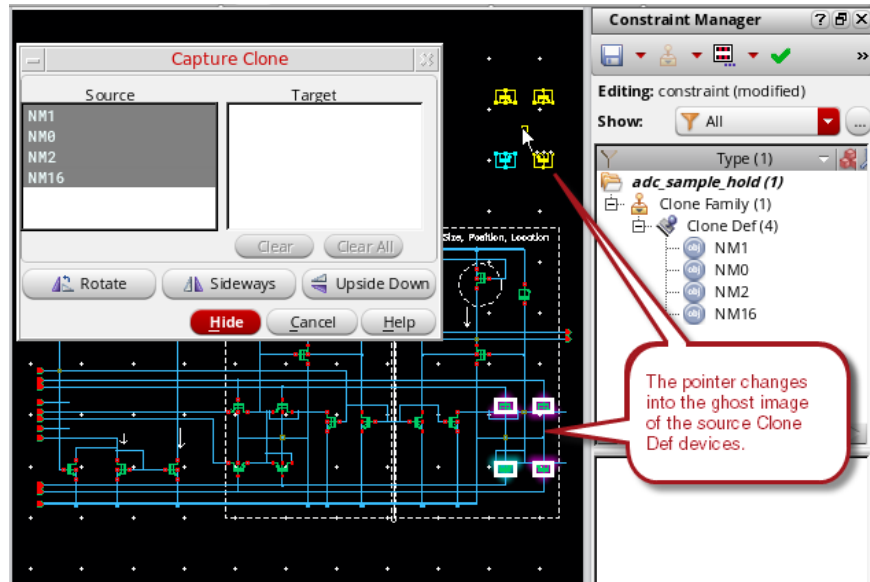
1. Right-click a *Clone Family* template or a *Clone Def* constraint. The context-sensitive menu displays the *Add Clone* option.
2. Select *Add Clone*.



This displays the Capture Clone form. In addition, the pointer changes into a ghost image of the source *Clone Def* devices, as shown below. When a *Clone Family* template is selected, the first *Clone Def* is used as source *Clone Def*. This pointer indicates a dragSet of the source *Clone Def*.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant



The Capture Clone form displays the following two list boxes and buttons to change the orientation of the dragSet:

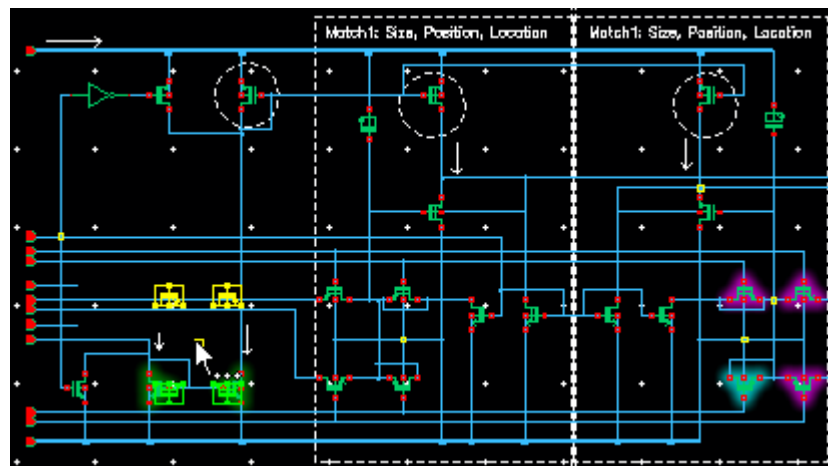
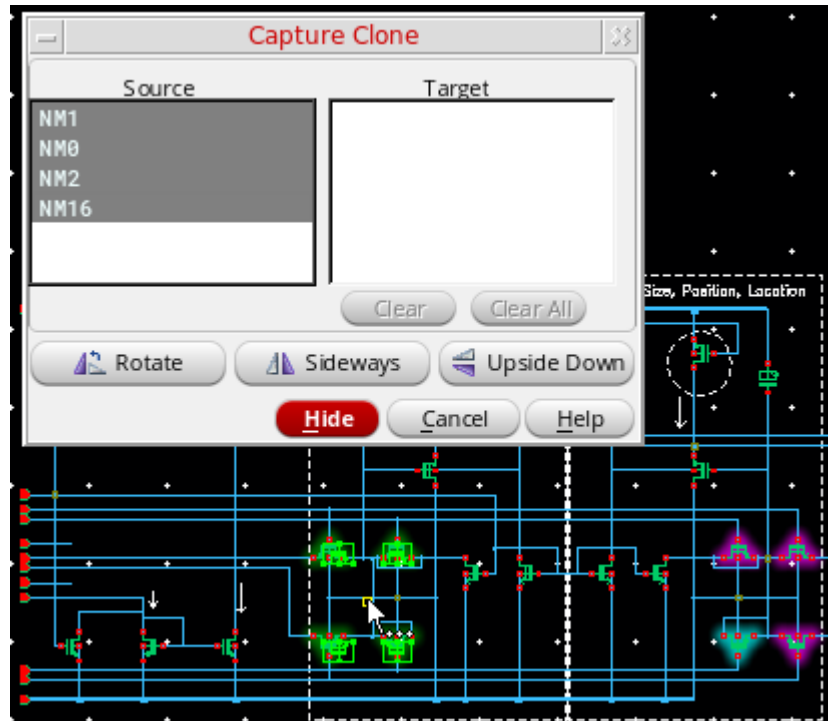
- ☐ *Source* lists the devices from the source *Clone Def*.
 - ☐ *Target* lists the devices that belong to the target *Clone Def*.
3. Move the pointer over target devices to identify whether they have the same topology as the source devices (the devices from the source *Clone Def*). If the target device has the same topology, it appears with green halo.

Note: For a target device to appear with green halo, it should have the same master as the source device. In addition, the target device should not already be a part of another *Clone Def* or *Modgen* constraint.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

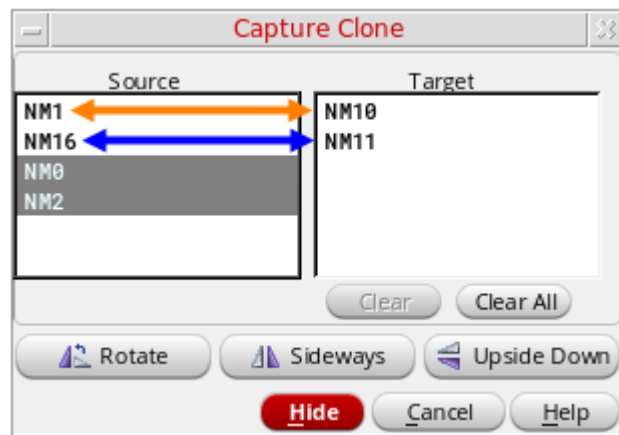
For example, the figures below show the difference in highlights based on pointer movement over different target devices. The first figure indicates all four target devices under the pointer have the same topology as the source devices; whereas, in the second figure, only two target devices out of four match.



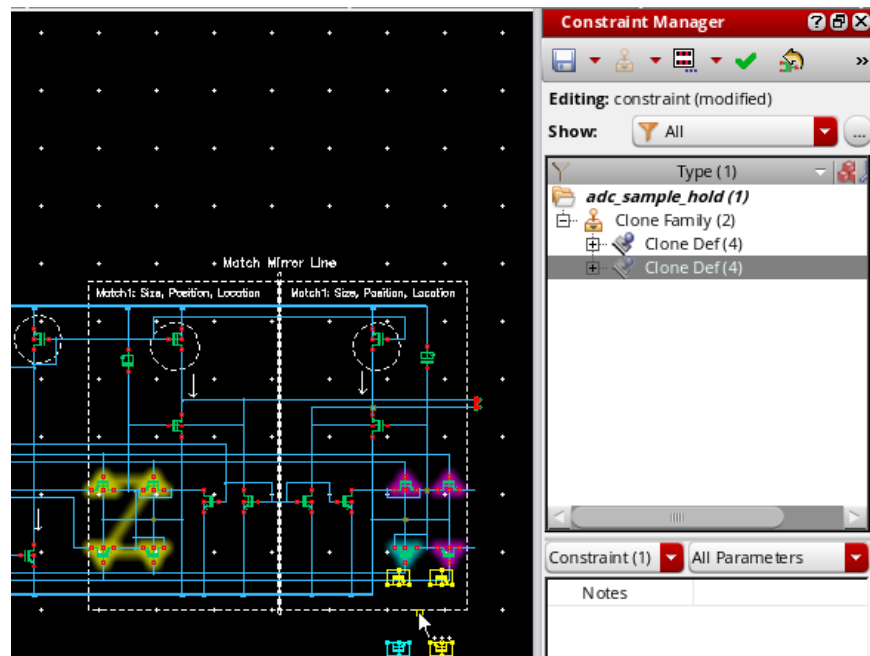
Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

- Click the target devices on which you want to create the new *Clone Def* constraint. The list boxes in the Capture Clone form show the correspondences between the source and target devices, as shown in the figure below.



After a matching target device has been selected for each source device, the new *Clone Def* constraint gets created and added to the *Clone Family* template, as shown below.



Important Points to Remember

While creating clones, you can:

- Remove devices from the dragSet by deselecting device names in the *Source* list box of the Capture Clone form or using `Ctrl+Click` or *Ctrl+Area select on source schematic devices*. The removed device gets highlighted with white halo and the pointer shows only the remaining source devices.
- Add (previously removed) devices from the source in the dragSet by selecting the device names in the *Source* list box of the Capture Clone form or using `Shift+Click` or *Shift+Area select on source schematic devices*.
- Remove a previously defined correspondence with `Ctrl+Click` or *Ctrl+Area select on a target schematic device*.
- Rotate the whole dragSet with center-click or `r` bindkey. You can also use the *Rotate* button in the Capture Clone form.
- Mirror X the whole dragSet with `Ctrl+Center-click`. You can also use the *Upside Down* button in the Capture Clone form.
- Mirror Y the whole dragSet with `Shift+Center-click`. You can also use the *Sideways* button in the Capture Clone form.
- Access the Capture Clone form from the Constraint Manager assistant in the schematic or layout. However, in both the cases, the functionality is run on the schematic window. Therefore, when this form is launched from the layout Constraint Manager, the layout cellview should be opened in Layout XL with a valid schematic “connectivity reference” cellview. In addition, all layout instances should be bound to schematic devices.
- Transfer from schematic to layout the entire *Clone Family* template as a complete unit.

Virtuoso Unified Custom Constraints User Guide

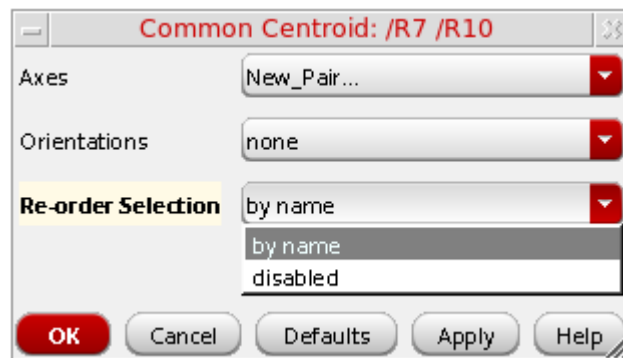
The Constraint Manager Assistant

- *Common Centroid* The *Common Centroid* template and constraint generator can be used to create a group of constraints that implements a common centroid (center of gravity) for ***two or four selected devices*** (see also Additional Use Case).

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

The order of devices selected is important. Therefore, if you have four devices currently selected either in the canvas or in the *Navigator*, choosing *Common Centroid* displays the Common Centroid form.



The Common Centroid form has the following options:

- **Axes** enables you to use a *New_Pair* of axes, or to select a pair of axes that is already in use by another *Common Centroid* template.
- **Orientations** enables you to choose the orientation constraint to be added to the constraints' template. The options here are: *none* (where no orientation constraint is created), and *vertical* and *horizontal* (where an orientation constraint is created based on the setting selected).
- **Re-order Selection** (*by name*): This option sorts the selected instances by name so that symmetries are created on the instances regardless of the selection order. For example, if instances $M2<1 : 0>$ and $M1<2 : 0>$ are selected, then:
 - ☐ The two horizontal symmetries are created between
 - ☐ $M1<0>$ and $M2<0>$
 - ☐ $M1<1>$ and $M2<1>$
 - ☐ The two vertical symmetry are created between
 - ☐ $M1<0>$ and $M1<1>$
 - ☐ $M2<0>$ and $M2<1>$

- **Re-order Selection** (*disabled*): This option disables the reordering of the selected instances, which allows you to create symmetries as per the order in which the instances are selected.

For example, if four devices are selected in the order of **A**, **B**, **C**, and then **D**, the *Common Centroid* template created contains three symmetries: the pairs **AB** and **CD** will be on the same vertical axis, and **AC** will be on a horizontal symmetry axis. An optional orientation with all four devices is also added, depending on the value of the *Orientations* option described above.

Note: Editing an orientation constraint within the template is prevented by the template callback, as is the deletion of any templates. Editing symmetry constraints is however allowed.

While creating a common centroid template, you need to first create the orientation constraint. If this constraint is created successfully, you need to continue with the symmetry constraints creation. Otherwise, you first need to find an existing orientation constraint with same members and parameters.

If you are able to find an existing orientation constraint with same members and parameters, you need to mark this constraint to be moved into the template, otherwise abort the template creation.

While creating the symmetry constraints, if any of the symmetry constraint cannot be created, then you need to delete the constraints that are already created for this template and abort the template. However, for symmetry constraints, you do not need to find an existing constraint. This is because the axes chosen are always unique or from the existing common centroids.

In addition, to avoid displaying of messages during constraint creation, you can create the constraints in the silent mode. This can be done by making the argument `verbose = nil` in `ciConCreate`. Once all the constraints are successfully created, you can show the creation messages at one go.

■ Using SKILL

A *Common Centroid* template can also be created in SKILL by using the template definition name "commonCentroid". For example, to create a Common Centroid template for devices A, B, C, and D, the command is as follows:

```
ciTemplateCreate cache "commonCentroid" ?members '(  
  ("A" 'inst) ("B" 'inst) ("C" 'inst) ("D" 'inst))
```

This creates a new template with two new axes.

Note: The ?userParams argument for ciTemplateCreate specifies the axes names and the optional orientation:

```
('("vertical" VAxisName_s) ("horizontal"  
HAxisName_s) ("orientation" "vertical" |  
"horizontal"))
```

The axes names and orientation can be specified in any order or combination, and are all optional. When an *Axis* parameter is missing, an axis is created, but when the *Orientation* direction is missing, the orientation constraint is not created. The calling function is responsible for creating and naming the axes and ensuring that the axes have the correct direction before it calls `ciTemplateCreate`.

■ Templates and Constraint Transfer

As is the case with all templates, any related constraints are fully transferable to the layout, and all transferred constraints form part of the resultant layout template.

■ Example of Use

If the *Common Centroid* template is created in a schematic, and the `mFactor` of all devices is identical after the devices are transferred to the layout template, the template contains 3x the `mFactor` value symmetries for each pair of `mFactor` instances.

In the example above, if `mFactor=2` for all devices, the layout template contains symmetries for the following pairs (A.1 B.1) (A.2 B.2) (C.1 D.1) (C.2 D.2) (A.1 C.1) (A.2 C.2).

If an orientation constraint exists in the schematic template, it will have all `mFactored` members in the layout transferred constraint. This is equivalent to having A.1 B.1 C.1 D.1 as a common centroid, and A.2 B.2 C.2 D.2 as another common centroid; the latter being a common centroid for each `mFactored` finger.

■ Additional Use Case

You can also create a *Common Centroid* template on two instances, given the condition that the `mFactor` (multiplier factor) on both instances is equal, and of an even number (at least 2).

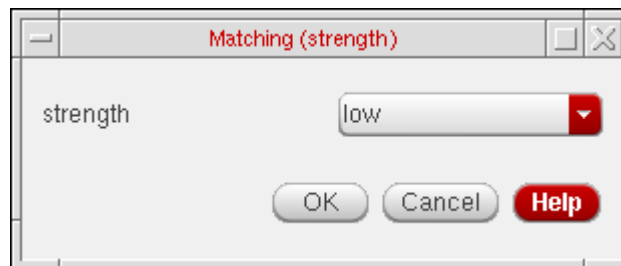
For example, a *Common Centroid* is created, in a schematic, on two instances (A and B) (both having `mfactor = 2`). Then, one *symmetry* constraint is created with A and B as members, and with a vertical axis. When this constraint is then transferred to layout, three symmetry constraints will be created on: (A.1-B.1) (B.2-A.2) (A.1-B.2). The first two constraints are on the same axis (vertical), as in the schematic, while the third constraint will be on the horizontal axis.

Similarly, when `mFactor` = 4, the *symmetry* constraint in the *Common Centroid*, in the schematic, would become 6 *symmetry* constraints in the *Common Centroid* template in layout. That is, for members A and B again, this would be $(A.1-B.1) (A.2-B.2) (A.1-B.2) (A.3-B.3) (A.4-B.4)$ and $(A.3-B.4)$. The third and sixth constraints would have a horizontal axis. So, the result is 1.5X constraints in layout where X is the `mFactor` value.

Important

Common Centroid is an example of a Constraint Template. Templates can be created and edited for various types of symmetries, alignments, and so on. For information on creating constraint generators, see Creating Custom Constraint Generators in the *Virtuoso Unified Custom Constraints Configuration Guide*.

- *Matching (strength)* *Matching (strength)* is an example template that can be used to generate constraints for objects that are currently selected in the design canvas. Selection will invoke the *Matching (strength)* form.

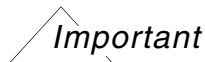


Using this form, you can set constraint matching *strength* to be either *low*, *medium*, or *high*.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

If matching constraints cannot be achieved, individual constraints inside the matching template (a managed collection of constraints) will be marked as *passed* (green).



Matching (strength) is an example of a Constraint Template. Templates can be created and edited for various types of symmetries, alignments, and so on. For information on creating constraint generators, see Creating Custom Constraint Generators in the *Virtuoso Unified Custom Constraints Configuration Guide*, and ciRunMatchingConstraintsGenerator SKILL function in the *Virtuoso Unified Custom Constraints SKILL Reference*.

- *Design Intent* If you have Virtuoso Design Intent enabled, this option will be available in Layout XL and lets you create a design intent using the *Constraint Manager*. For more information on design intents, refer to the *Virtuoso Design Intent User Guide*.

Constraint Editors (schematics and layout)

Drop-down menu that lists available constraint editing functionality: *Module Generator...*, and the *Process Rules Editor...*. Selecting any of these options will launch the associated application.

Whatever constraint editor was selected last will be shown as the “default” option to be selected for next use.

Note: The constraint editor options remain active even when the current cellview is in read-only mode.

- *Module Generator...* Invokes the layout editor and displays the Modgen Editor.
The Module Generator provides a simple and fast method of generating multiple Pcell instances into a complex, highly matched array based on defined Pcell leaf structures.
For more information, see Editing Modgens in the *Virtuoso Module Generator User Guide*.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

- *Process Rule Editor...*

Invokes the Process Rule Editor form.

From here you can edit process rules for the technology file, the design, or only for selected objects. Alternatively, you can create new constraint groups or process rules.

For more information, see [The Process Rule Editor](#).

Check Constraints (Schematics)

When you click *Check Constraints* from the Constraint Manager assistant of the Schematic window, if the cellview contains constraints the [Constraint Status](#) and [Template Status](#) check starts automatically based on the current settings in the *Constraints* tab of the Schematic Rules Check Setup form. Alternatively, you can use the [ciCheckConstraints](#) SKILL function to run the constraint checker.

Note: To access the Schematic Rules Check Setup form, select *Check – Rules Setup* in VSE. For more information, see [Constraints Checks](#) in the [Virtuoso Schematic Editor User Guide](#).

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Check Constraints (Layout)

When you click *Check Constraints* from the Constraint Manager assistant of the Layout window, PVS-CV gets launched to check whether the specified constraints have been implemented correctly in the layout. Alternatively, you can use the [ciCheckConstraints](#) SKILL function to run the constraint checker.

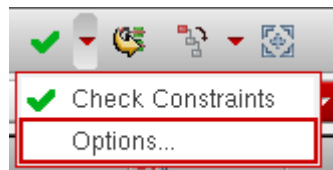
After the checks, the status of the constraints gets updated in the Constraint Manager assistant as any one of the following: *Passed, Failed, or Not Checked*.

In addition, any constraint violation detected during this check by PVS-CV is reported in the Annotation Browser assistant of the Layout window.

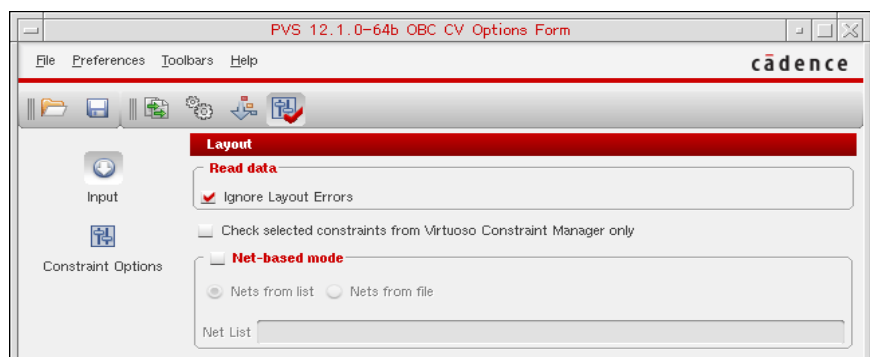
PVS-CV is a licensed tool. Therefore, to use the *Check Constraints* option from the Constraint Manager assistant of the Layout window, the

Phys_Ver_Sys_Const_Validator (license number 96300) license must be checked out in addition to the Layout license. Initially, PVS-CV runs on the layout with default settings. However, you can change these settings using the PVS-CV form before running the Constraint Checker.

To open the PVS-CV form, in the Constraint Manager assistant, select *Options* from the *Constraints Check* drop-down list.



The PVS-CV Options Form is displayed, as shown below.



The options for running PVS-CV from Constraint Manager is taken from the `.cadence/pvs` directory. If you are running PVS-CV for the first time, this information is created in the `.cadence/pvs` directory.

The PVS-CV Options Form contains the Input and Constraint Options panels represented by their icons with appropriate names.

Input

When you open the PVS-CV form, the *Input – Layout* section is displayed by default. You can set the following fields in the *Layout* section:

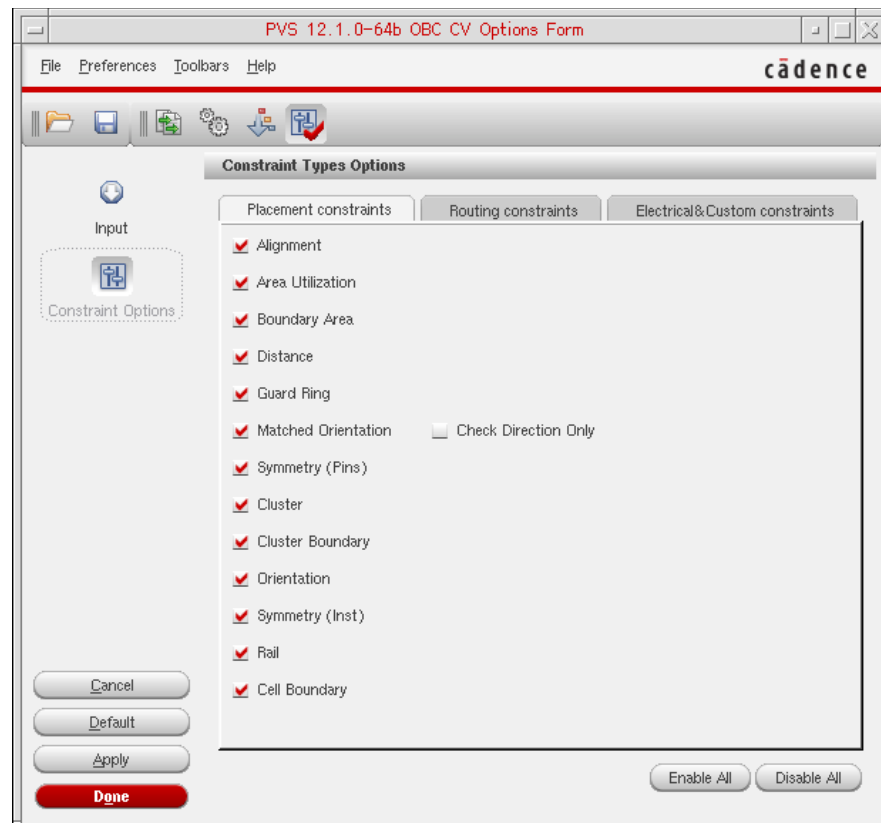
- *Read data*
 - ☐ *Ignore Layout Errors* — Select this check box to ignore an input design error (for example, some master cells may be absent). By default, CV detects layout errors and terminates the run.
- *Check selected constraints from Virtuoso Constraint Manager only* — Select this option to check only the constraints selected in the Constraint Manager.
- *Net-based mode* — When selected, only constraints on specified nets are checked.
 - ☐ *Nets from list* — Net names are typed into the text field.
 - ☐ *Nets from file* — Name of a text file is entered in the text field.

Constraint Options

- ☐ When you click the *Constraint Options* icon, the *Constraint Types Options* section is displayed.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant



The Constraint Options panel consists of three tabs related to different groups of constraint types.

- The *Placement constraints* tab displays the list of all supported placement constraints in Virtuoso, such as Alignment, Area Utilization, Boundary area, Distance, Guard Ring, Matched Orientation, Symmetry (Pins), Cluster, Cluster Boundary, Orientation, Symmetry (Inst), Rail, and Cell Boundary.
- The *Routing constraints* tab displays the list of all supported routing constraints in Virtuoso, such as Shielding, Diff Pair, Bus, Matched Length, Symmetry (Nets), PR Overrides, Net Class.
- The *Electrical&Custom constraints* tab contains set of Electrical constraints and Custom constraints.

The Constraint Options panel is used to set up constraint types that you want to check.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

To enable or disable some specific constraint types, just click the appropriate check box against the constraint name.

- *Enable all* — If you want to check all or most of constraint types, use this button at the lower-right corner of the form. It makes all constraint types enabled and you can deselect the ones you do not want to check.
- *Disable all* — If you want to check only a few of constraint types, use this button at the lower-right corner of the form. It makes all constraints disabled and you can select the ones you want to check.

For more information on running PVS-CV, see the [*Running PVS CV*](#) chapter in the *Cadence Physical Verification User Guide*.

Also, for detailed description on PVS-CV functionality, see [*Appendix H- PVS CV Functionality*](#) in the *Cadence Physical Verification User Guide*.

Constraint Comparison Report (Layout only)

Invokes a web page that compares the constraints set on the schematic and layout views, sorting any differences into various categories.

Each constraint is detailed in the report with each change that has been made to its members, parameters, or attributes being flagged.

For more information, see [Comparing Constraints between Schematic and Layout](#).



Video

For a video demonstration of the process of generating a constraint comparison report, see [Comparing Constraints Between Schematic and Layout](#) on Cadence Online Support.

Update Schematic Constraints (Schematics only)

Performs a constraint status update for the existing schematic constraints against the equivalent constraint status in layout.

For more information, see [Updating Constraints and Constraint Status](#).

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

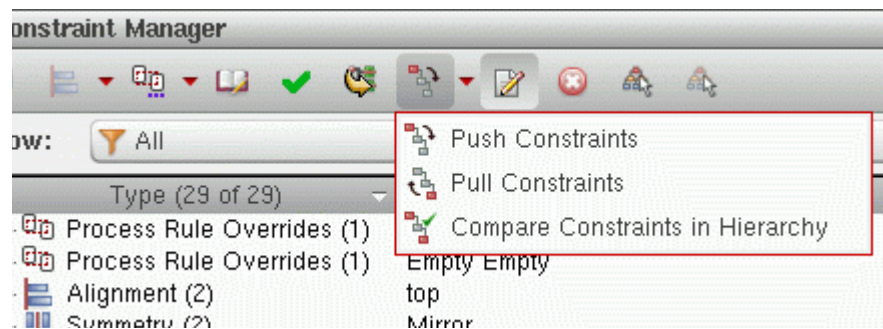
Update Selected Layout Constraints From Schematic (Layout only)

Performs a constraint status update for the existing layout constraints against the equivalent constraint status in schematics.

For more information, see [Updating Constraints From Schematic to Layout](#).

Constraint Propagation (Schematics and Layout)

Contains options to propagate constraints through the layout hierarchy.



For more information, see [Pushing and Pulling Constraints in a Schematic or Layout Hierarchy](#)

Show Selected (Schematics and Layout)

Toggles on and off the display of the selected constraints in the Constraint Manager assistant.

This feature helps you in identifying only those constraints that are currently selected in the design canvas. This can be especially useful if you have a high number of constraints listed in the *Constraint Manager*.

Delete (Schematics and Layout)

Deletes the currently selected constraint (or object group) in the *Constraint browser*.

Select All (Schematics and Layout)

Selects all constraints in the *Constraint Browser*.

Deselect All (Schematics and Layout)

De-selects all constraints in the *Constraint Browser*.

Constraint Parameter Editor (Schematics and Layout)

Toggles on and off the display of the parameter editing pane.

Note: The parameter editing table can also be displayed by double-clicking a constraint in the *Constraint Browser*.

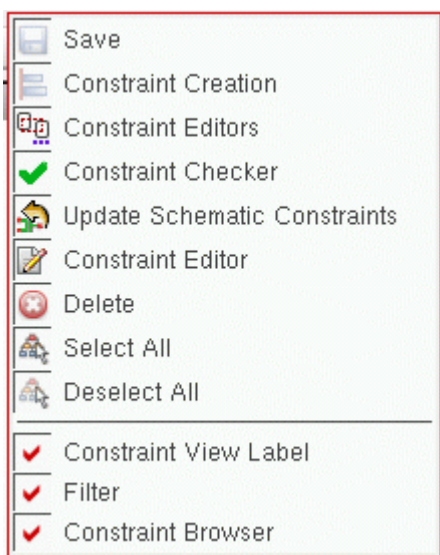
Customizing Constraint Manager Assistant Content

You can customize the options on display in the *Constraint Manager* toolbar, and the Constraint Manager assistant itself.

Note: Additional customizations are detailed throughout this document.

To add/remove *Constraint Manager* user-interface options and feature content:

1. Right-click over the *Constraint Manager* toolbar area. This displays a context-sensitive menu that provides options for customizing the interface. For example, the figure below shows the context-sensitive menu displayed in Schematic XL.



2. Select/deselect the elements that you want to have on display in Constraint Manager assistant.

Saving and Restoring Constraint Manager Customizations

All *Constraint Manager* user-interface customizations are saved to, and restored from, setting files that are stored in the `.cadence/dfII/ci` directory.

Note: There are separate settings files for the schematic constraint manager and the layout constraint manager to allow for separate configurations.

All settings found in the Cadence Search Path, as specified in `setup.loc`, will be loaded in order. This allows for default settings files to be stored under `share/cdssetup/dfII/ci` and for CAD departments to specify their own customizations.

Module Generator and Process Rule Editor Constraint Manager Options

You can run constraint editing functionality, such as layout cell planning and modgen functionality, by selecting *Module Generator...*, or *Process Rule Editor...* from the *Constraint Editor* menu on the *Constraint Manager Toolbar*. Alternatively, you can open an external constraint editor by double-clicking a constraint in the *Constraint Browser*. Use the 'constraintType' argument of the *ciRegisterConstraintEditor* SKILL function to specify the constraint type that can be edited using a particular constraint editor. In addition, you can configure to open an external constraint editor when you double-click a template in the *Constraint Browser* or click any template parameter in the *Constraint Parameter Editor* pane. For detailed information, refer to [Defining the Interface for Editing a Template and its Parameters](#).

The constraint editor that was selected last is shown as the “default” option to be selected for next use.

If these buttons are available, they will be displayed as either enabled or disabled (grayed out) as follows:

- Both buttons are enabled if there is nothing selected in the *Constraint Manager*. Selecting either button will mean that you want to create a new ModGen or CellPlan.
- Both buttons are disabled if you select a constraint that is not a ModGen or CellPlan.
 - If one button is enabled, and one disabled, this indicates that you have selected a ModGen or CellPlan.

For more specific information on physical design constraints and their uses in Layout XL, see [Constraint Manager in Layout XL](#). See also [Editing Modgens in Virtuoso Module Generator User Guide](#).

For information on how to plug in other third party constraint editing tools, see [Custom Constraint Editors in Virtuoso Unified Custom Constraints Configuration Guide](#).

For more information on the *Process Rule Editor*, see [The Process Rule Editor](#)

Updating Constraints

The *Update Schematic/Layout Constraints* options, found on the respective application *Constraint Manager* toolbars, are used to synchronize the constraints between the two views. The status of the constraints (see [Constraint Status](#)) in one view, with respect to the other view, is captured in the status column along with information as to whether or not the constraints are, for example, enforced or violated.

The enablement of the *Update Schematic/Layout Constraints* options and the content of the icon tooltip is dependent upon the current setting of the `updateConstraints` environment variable and whether or not any constraints are currently selected. Also, see [Update Schematic/Layout Constraints Options Tooltips](#) for related information.

Constraints in the target view will be updated if the corresponding constraints in the source view are more recent. This will only apply when you choose to update *all* constraints.

Note: Related to this, selecting the [Check](#) option on the *Constraint Manager* toolbar automatically performs status checks based on the current settings in the *Constraints* tab of the Schematic Rules Check Setup form (for schematics) or on the current settings in the Batch Checker form (for layout).

Only those constraints that exist in both schematics and layout can have their status updated in the schematic *Constraint Manager*.

When the equivalent layout cellview is open, the schematic *Constraint Manager* will automatically update the latest layout status whenever you switch between schematic or layout view tabs. The *Update Schematic Constraints* command ensures that the constraints in the schematic are in sync with the layout, by pulling in all/selected constraints in the layout into the constraint view, and updating the status information displayed in the schematic *Constraint Manager*.

Note: Although this command will sync any design intent changes if Virtuoso Design Intent is enabled, its primary function is to update constraints. To sync design intent changes, the *Sync All Design Intent* command should be used instead. For more details, see the [Virtuoso Design Intent User Guide](#).



Figure 1-4 Update Schematic Constraints option on the Constraint Manager toolbar

To update constraints in the schematic *Constraint Manager* (constraint view) and reflect constraint status in the layout:

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

1. With the Constraint Manager assistant on display in a schematic view, launch the layout tool (for example *Launch – Layout XL*).
2. If you now open the equivalent layout cellview, the layout *Constraint Manager* will automatically display inherited schematic constraints on any corresponding components and nets, in addition to any constraints that were entered directly in the layout.

Note: A blank layout will not show any constraints.

Note: See [Transferring Constraints Between Schematics and Layout](#) which describes how schematic constraints are transferred to layout.

3. Clicking the *Update Schematic Constraints* option in the schematic *Constraint Manager* toolbar brings in the constraints in layout that do not currently exist in the schematic, and syncs up constraints that are different (have different parameters) between schematic and layout. In addition, the status of the constraints in the schematic are also updated.

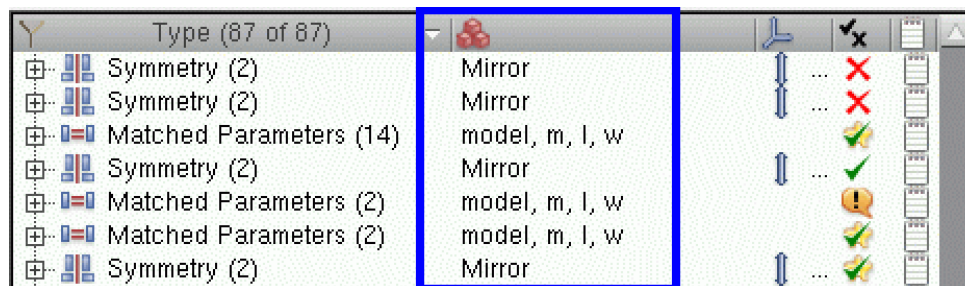


Figure 1-5 The Status column in the schematic Constraint Manager

Important

Currently, only selected constraints will be updated using this process, and not selecting any constraints will serve as an “update all” operation.

- ❑ For more information on the constraint status, see [Constraint Status](#).
- ❑ For information on how to access a constraint comparison report, between schematic and layout, see [Comparing Constraints between Schematic and Layout](#).

Controlling Constraint Updates Using the updateConstraints Environment Variable

The use model of the *Update Schematic/Layout Constraints* options can be controlled using the [updateConstraints](#) environment variable.

Update Schematic/Layout Constraints Options Tooltips

The content of the *Update Schematic/Layout Constraints tooltip* is dependent upon the current setting of the `updateConstraints` environment variable and whether any constraints are selected.

The tooltips can therefore be one of the following, for example:

- *Update all schematic constraints from layout.*
- *Update selected schematic constraints from layout.*
- *Update all layout constraints from schematic.*
- *Update selected layout constraints from schematic.*
- *Cannot update schematic constraints because the corresponding layout is not open.*
- *Cannot update schematic constraints because "overview block1 constraint" is read only.*
- *Cannot update layout constraints because "overview block1 layout" is read only.*

Editing a Constraint in Associated Constraint Generator Dialog Box

To edit a constraint, open the associated *Constraint Generator* dialog box by a double-clicking one of the following:

- The constraint template in the Constraint Manager assistant
- Any of the constraint parameter value in the Constraint Manager assistant's property editor pane

For an example, see [Editing a Generated Modgen \(GenericModgen Template\)](#).

Constraint Browser

The *Constraint Manager* displays design constraints (in creation order), along with their parameters, in the *Constraint Browser*. You can use the *Constraint Manager* to browse, add, modify, and delete constraints.

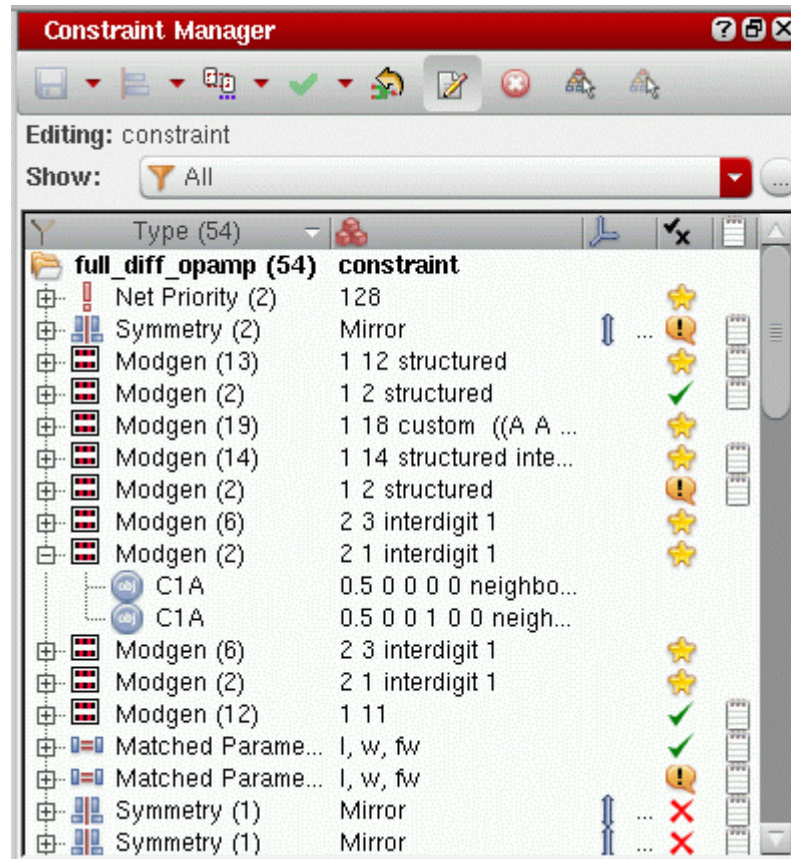


Figure 1-6 The Constraint Browser (schematics)

Each constraint is displayed in its own row, with the browser itself comprising a selection of columns and cells that display properties associated with each constraint. Icons and variable colors and fonts are used to aid parameter visualization. This sections discusses:

- [Constraint Filters](#)
- [Constraint Browser Features](#)
- [Constraint Browser Context-Menus](#)
- [Constraint Browser Columns](#)

- ❑ [Constraint Types](#)
- ❑ [Constraint Parameters](#)
- ❑ [Constraint Axis](#)
- ❑ [Constraint Status](#) (see also [Updating Constraints](#))
- ❑ [Constraint Notes](#)

Constraint Filters

Filters are used in the *Constraint Manager* to restrict the constraint types and members displayed to be only those that match certain, filtered, criteria.

For example, you can choose to only display those constraints of type *Alignment* and/or *Correlation*. Alternatively, you can choose to display all constraint types, but only show constraint members of a certain type, for example only nets and pins.

Constraint filter options are accessible from the *Show* pull-down on the Constraint Manager assistant.

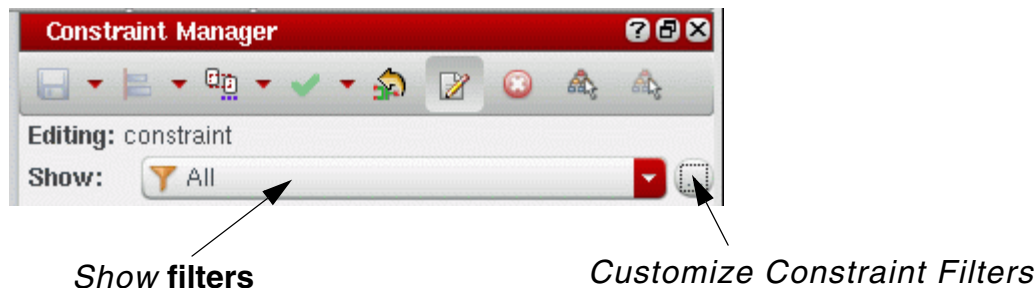


Figure 1-7 Constraint Filter options on the Constraint Manager

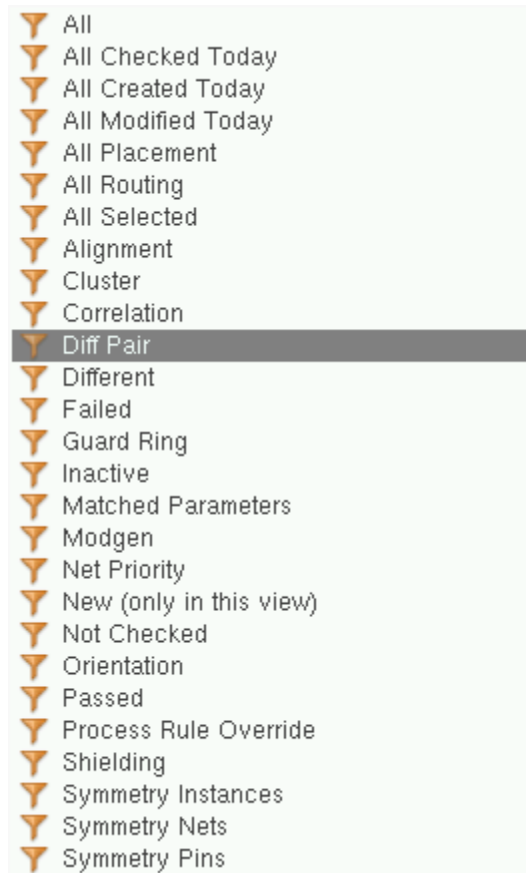


Figure 1-8 Filters available from the Show pull-down on the Constraint Manager

Alongside any filters that you have created manually (see [Creating a Constraint Filter](#)), the *Show* drop-down also includes a list of (editable) template filters. In addition to providing the ability to filter on particular constraint types, these pre-defined constraint filters let you choose to filter constraints based on their timestamp/history information (for example *All Created Today* or *All Modified Today*) or to only view those constraints that are associated with placement or routing (*All Placement* or *All Routing*). By default, *All* constraints (types and members) will be displayed in the *Constraint Manager*.

Note: Choose the *All Selected* filter if you are only interested in identifying those constraints that belong to any objects (constraint members) that are currently selected in the design canvas. This can be especially useful if you have a high number of constraints listed in the *Constraint Manager*.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Applying a Constraint Filter

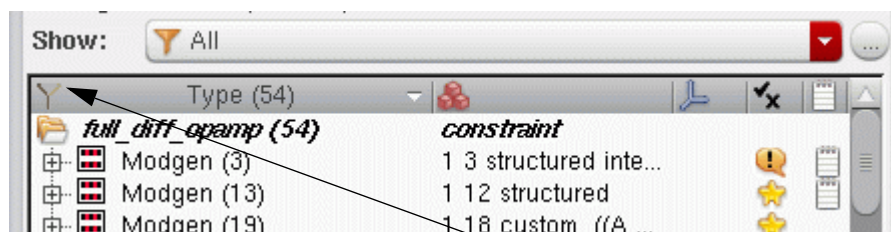
To apply a constraint filter:

- Select the constraint filter that you want to apply from the *Show* pull-down in the *Constraint Manager*.

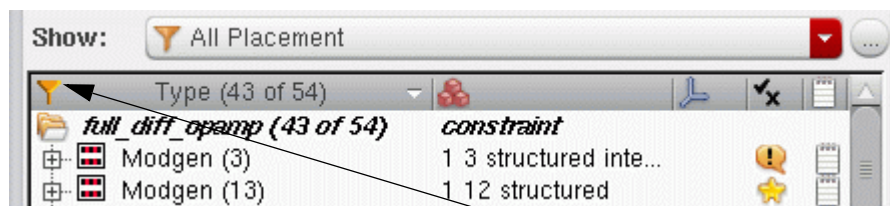
The *Constraint Manager* display will update to show only those constraint types and members that are associated with the applied constraint filter.

Note: An applied filter will also take account of any constraints contained within Constraint Templates.

When a filter has been applied, the filter icon in the *Constraint Browser* will show as filled (in orange). Floating your cursor over the *Type* column header will also display what type of filter is currently applied.



No filter currently applied



Filter currently applied

Creating a Constraint Filter

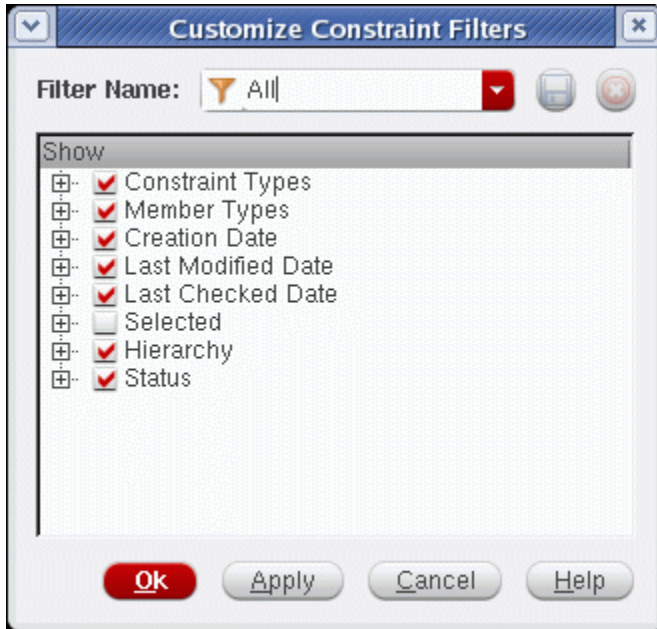
To create a constraint filter:

1. Select the ... (*Customize Constraint Filters*) option adjacent to the *Show* pull-down.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

This will display the Customize Constraint Filters form.



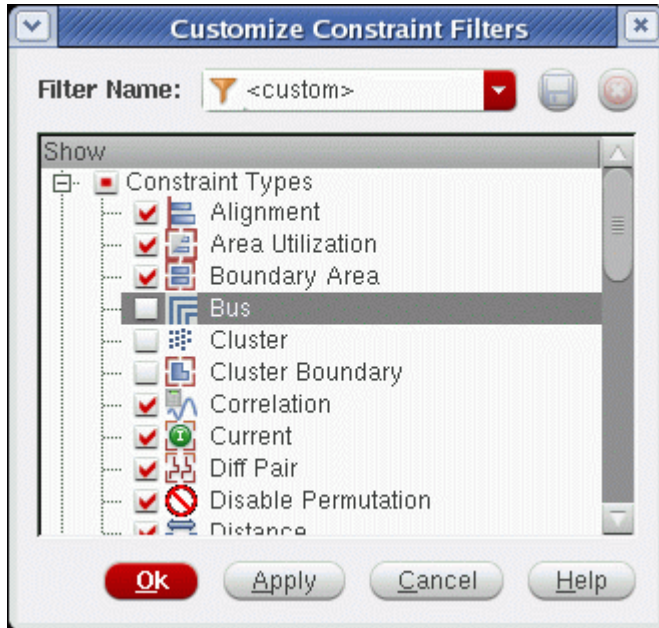
2. In the *Filter Name* field, overwrite any current filter name and enter the name of the new constraint filter to be created.

Note: You should only use A-Z, a-z, and 0-9 characters when creating a new filter name.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

3. In the *Show* section, select/deselect the *Constraint Types* that are to be shown (included) when the new constraint filter is selected in the *Constraint Manager*.

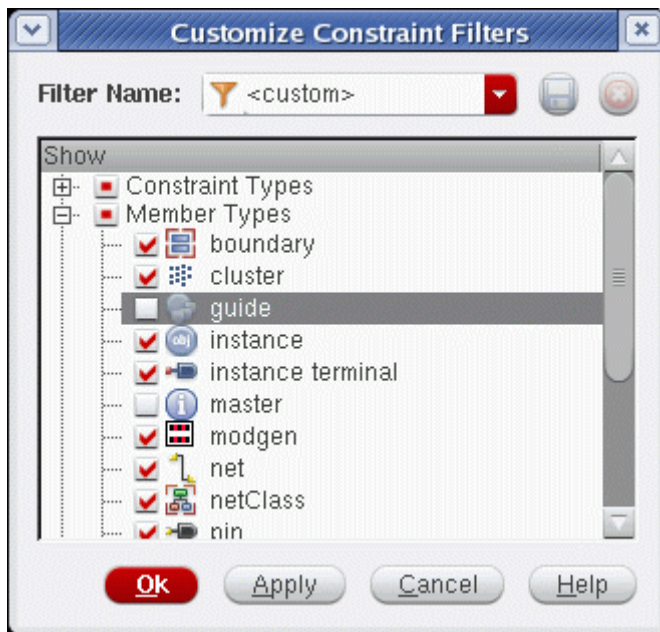


Note: If you did not enter a new filter name by this stage, the *Filter Name* field will display *<custom>* once you start to customize the content of the, as yet, unnamed filter. A *<custom>* filter can be applied (by selecting the *Apply* button), but it cannot be saved (the *Save* icon at the top of the form will be inactive).

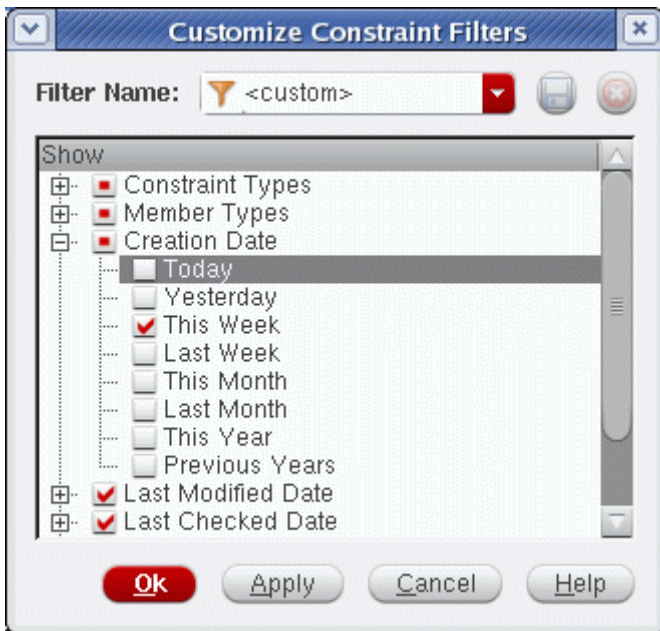
Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

4. Optionally, in the *Show* section, select/deselect the *Member Types* that are to be shown (included) when the new constraint filter is selected in the *Constraint Manager*.



5. Optionally, in the *Show* section, choose any history attributes that you would like to filter on using the *Creation Date*, *Last Modified Date*, and *Last Checked Date* options.



For more information on constraint history attributes, see [Constraint History](#).

6. Optionally, in the *Show* section, choose to only apply the new constraint filter to the objects currently selected in the design canvas (or in the *Constraint Manager*) by checking the *Selected* check box.

Note: Checking the *Selected objects only* option will automatically check the *Selected* check box.

7. Optionally, in the *Show* section choose to filter on constraint *Hierarchy* filter settings.

Uncheck the *Hierarchy* check box to hide all constraint sub-directories in the *Constraint Manager*.

Check the *Hierarchy* check box to show constraint sub-directories in the *Constraint Manager*. The options contained in the *Hierarchy* filter now allow you to control the display of overridden constraints in different levels of the hierarchy.

- ☐ Choosing to filter on *Overridden sub-hierarchy* will only show, in the *Constraint Manager*, those constraint sub-directories that contain overridden constraints.
- ☐ Choosing to filter on *Un-overridden sub-hierarchy* will only show constraint sub-directories that do not contain overridden constraints.

For more information see [Creating Constraints at Different Design Levels](#) and [Overriding Constraints](#).

8. Select the *Save* icon at the top of the form to store the new constraint filter.

The new constraint filter will now be available for selection from both the *Show* and *Filter Names* pull-downs for any cellviews opened.

Note: Filters are stored in the `.cadence/dfII/ci/filters` directory as SKILL files (.il). If there are any problems when filters are being re-loaded a warning message will be displayed in the CIW.

Updating a Constraint Filter

To update a constraint filter (user created or template):

1. Select the ... (*Customize Constraint Filters*) option to the right of the *Show* pull-down.
This will display the Customize Constraint Filters form.
2. Select the constraint filter to be updated from the *Filter Name* pull-down.
3. Optionally, in the *Show* section, select/deselect the *Constraint Types* that should now be shown (included) in the updated constraint filter.

4. Optionally, in the *Show* section, select/deselect the *Member Types* that should now be shown (included) in the updated constraint filter.
5. Optionally, in the *Show* section, check/uncheck the *Selected* check box to determine if the filter should only be applied to the objects currently selected in the *Constraint Manager*.
6. Select the *Save* icon at the top of the form to update the currently selected filter.
You will be prompted to overwrite the currently selected filter.
7. Select *Yes* in the *Save Filter* form to update the filter.
The updated filter will now be available for selection from both the *Show* and *Filter Names* pull-downs.

Deleting a Constraint Filter

To delete a constraint filter (user created or template):

1. Select the ... (*Customize Constraint Filters*) option to the right of the *Show* pull-down to display the *Customize Constraint Filters* form.
2. Select the constraint filter to be deleted from the *Filter Name* pull-down.
3. Select the adjacent *Delete Filter* button.
You will be asked to confirm the deletion of the selected constraint filter.
4. Select *Yes* in the *Delete Filter* form to confirm the deletion of the selected constraint filter.
The constraint filter will now be deleted from both the *Show* and *Filter Names* pull-downs and the (show) *All* (constraints) filter will automatically be applied.

Note: If you are having difficulties deleting a filter you can remove it directly from `.cadence/dfIII/ci/filters`.

Constraint Browser Features

You can use the *Constraint Browser* to:

■ **Sort columns**

You can perform an alphabetical sort on any column by clicking the column header.

Note: By default, constraints are displayed by creation date.

■ **Customize browser display**

You can choose what information you want displayed, and how it is displayed, by right-clicking over the column header area of the *Constraint Browser*. From here, you can set your preferred sort order, displayed columns, and so on.

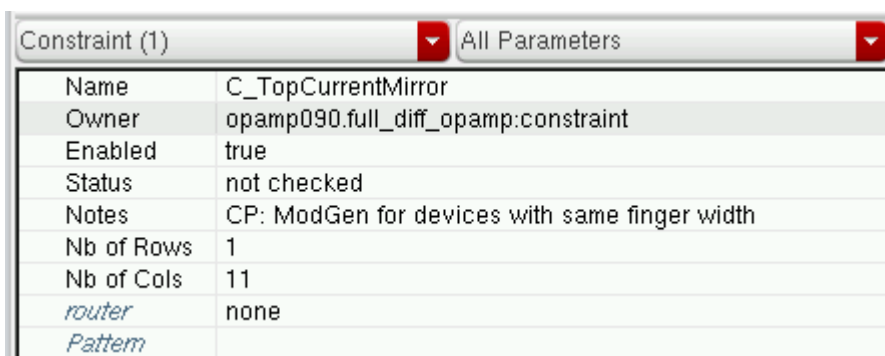
For more information, see [Constraint Browser Column Header Context-Menu](#).

■ **Handle in-context and out-of-context constraints**

The *Constraint Manager* displays whether a constraint is currently *in-context* or *out-of-context*. That is, the constraint is in the same hierarchy, but as part of a different traversal.

Constraints that have a white background are constraints that were created in the current view (currently in-context), while those constraints with a light gray background were created in another view (currently out-of-context).

When a constraint is out-of-context, you can see where it was originally created by looking at the *Owner* constraint parameter in [Constraint Parameter Editor](#).



Constraint (1)		All Parameters
Name	C_TopCurrentMirror	
Owner	opamp090.full_diff_opamp:constraint	
Enabled	true	
Status	not checked	
Notes	CP: ModGen for devices with same finger width	
Nb of Rows	1	
Nb of Cols	11	
router	none	
Pattern		

All constraints are visible in schematic and layout. There are two exceptions for layout:

- ❑ If an out-of-context constraint has hierarchical members and the `CI_ALLOW_LAYOUT_HIER_OCC` shell environment variable is not set, the constraint will not be visible.

- ❑ If a constraint is in-context, but has members of type *modgen* or *cluster* that are not visible because their members are hierarchical and the `CI_ALLOW_LAYOUT_HIER_OCC` shell environment variable is not set.

You can transfer the following types of constraints:

- ❑ All in-context constraints are transferable in schematic and layout except when a constraint is in-context and its members are themselves an out-of context constraint. For example, there is a *modgen* or *cluster* constraint which is out-of-context. If such a constraint is a member of another constraint, the parent constraint will not be transferred because the child constraint is out-of-context.
- ❑ All out-of-context constraints in layout are transferable if their members are hierarchical and the `CI_ALLOW_LAYOUT_HIER_OCC` shell environment variable is set.

■ **Reorder columns**

You can re-position a column by clicking and dragging the column header and dropping the column header in its new position.

■ Drag-and-drop one constraint to add as a member of another constraint

Existing constraints can be dragged and dropped onto another constraint as long as that constraint can support the dragged constraint as a member.

For example, this allows *Modgen* constraints to be dragged and dropped into *Symmetry* constraints, *Alignment* constraints, and so on. It also allows for *Cluster* constraints to be dragged and dropped into other *Cluster* constraints, as well as enabling a *Cluster* constraint to be dragged and dropped onto a *Cluster Boundary* constraint (replacing the *Cluster* that was associated with that *Cluster Boundary*).

Note: Dragging a constraint, and dropping it into another constraint, will **add** that dragged constraint as a member of the other constraint. However, dragging a constraint and dropping it onto a constraint *member* will replace the existing member with the dragged constraint.

■ **Display constraint type members**

You can view all of a constraint type's members by expanding one or more of the constraint *Type* branches.

To do this, click the horizontal arrow icon to the left of the constraint *Type*. Once expanded, all members (*Inst*, *Net*, and so on) will be displayed under the constraint type, for example all of instances or nets of constraint type *Alignment* or *Orientation*

can be viewed. The arrow icon changes to be positioned vertically, and clicking again on this icon will collapse the view, hiding all constraint type members.

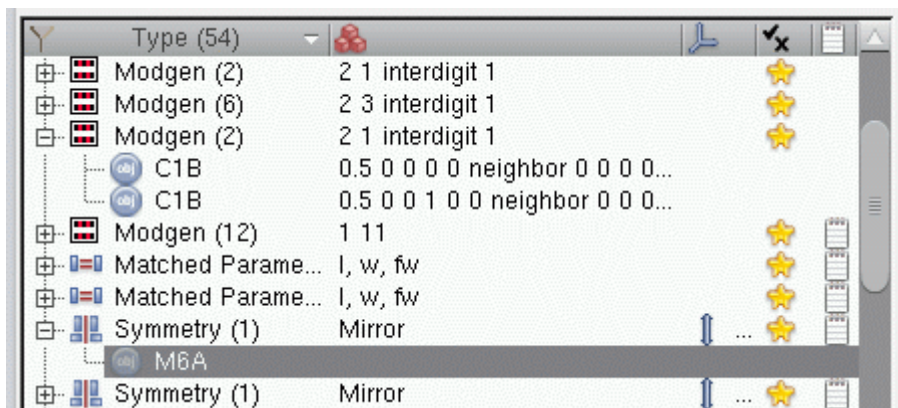


Figure 1-9 Some expanded constraints displaying their members

You can rearrange a constraint's member order using drag-and-drop (although order is not of importance in most constraints). When commencing a drag-and-drop, the cursor will change to reflect valid and invalid drop points. This rearrangement of the member order automatically triggers consistency check on the constraint.

Note: Constraint member details can be drag and dropped into applications that accept plain text, including the [Search assistant](#) (allowing you to quickly run searches based on the selected constraint member details).

■ **Highlight constraints on the canvas**

You can select a constraint in the browser which causes the constraint members to be highlighted on the canvas using *halos*. For more information on halos, see [Halo Highlighting](#).

Note: If you search on constraints using the [Search assistant](#), selecting one or more constraint results in the *Search assistant* will cause the constraints to be automatically selected in the *Constraint Manager*. Double-clicking these results will also run the default action of selecting all constraint members and having these selections also cross-selected in the *Navigator* and *Property Editor* assistants.

■ **Display constraint member parameters in the Constraint Parameter Editor**

Whenever a selection is made in the *Constraint Browser*, [Constraint Parameter Editor](#) will update (assuming that it is activated by double-clicking a constraint or by ensuring that the *Constraint Parameter Editor* option is selected in the Constraint Manager assistant toolbar) and present the associated parameters and parameter values of the selected constraints. If you select multiple constraints from the browser, you will be able to edit multiple constraints at the same time in the *Constraint Parameter Editor* pane.

Standard `Shift-select` and `Control-select` is available for selection of multiple items in the browser.

For more information on selecting in the *Constraint Browser*, see [Selection and Cross-Selection of Constraints](#).

■ **Cross selection with the Navigator**

Selections made in the *Constraint Browser* will be reflected in the *Navigator*, and vice-versa.

For more information, see [Selection and Cross-Selection of Constraints](#) and the *Navigator Assistant* in the [Virtuoso Schematic Editor User Guide](#).

■ **Resize/Re-Position columns**

You can resize a column by clicking and dragging on the column header vertical sides. Double-clicking the vertical separator in any column header will also auto-size column width. You can re-position a column by clicking and dragging it to a new location.

■ **Tooltips**

Tooltips are displayed for each constraint listed in the *Constraint Manager* by hovering the mouse cursor on a specific constraint.

These tooltips include:

- ❑ For the constraint directory in the current constraint view (see [The Constraint View](#)) - information on the library, cell, and view names, and the number of constraints in the current view.

Note: Information on the number of constraints in the current view is also displayed in the *Type* column for the constraint directory.

- ❑ For individual constraints - information on the default constraint name, the constraint type, its ownership setting, its constraint member information, any override information.

Note: Constraint member information could include, for example, what the left and right members are for a symmetry constraint.

Note: Tooltip information is also available in the *Notes* column by moving the mouse cursor over the note icon.

■ **Use toolbar shortcuts**

The *Constraint Browser* contains a *Constraints* toolbar which provides shortcuts that allows you to quickly add and delete constraints, create or remove templates, and so on. For more information, see [Constraint Types](#).

■ **Use context-menu options**

Right click over the *Constraint Manager* assistant to display a related context-menu.
For more information see [Constraint Browser Context-Menus](#).

Key Bindings

You can use the following bindkeys when using the *Constraint Manager*:

Key Binding	Action
Delete	Deletes the current selection.
Ctrl + A	Selects all constraints.
Right Arrow	Expands the current constraint to show members.
Left Arrow	Collapses the current constraint to hide members.
Up/Down Arrows	Move up and down the <i>Constraint Manager</i> tree.
Shift/Ctrl + Select	Used for extended selections.

Constraint Browser Context-Menus

Depending on where you right-click in the *Constraint Browser*, a different context-menu will be displayed.

Constraint Browser Column Header Context-Menu

If you right-click over the *Constraint Browser* column header area, the following context-menu is displayed:

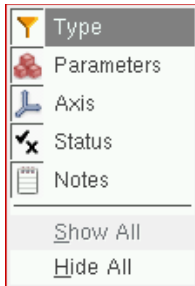


From here:

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

- Selecting the *Columns* sub-context-menu displays the following menu:



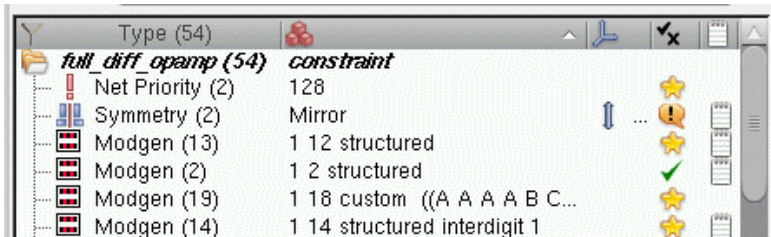
From here you can choose which table column headers that you want to display in the *Constraint Browser*.

- Selecting the *View By* sub-context-menu displays the following menu:

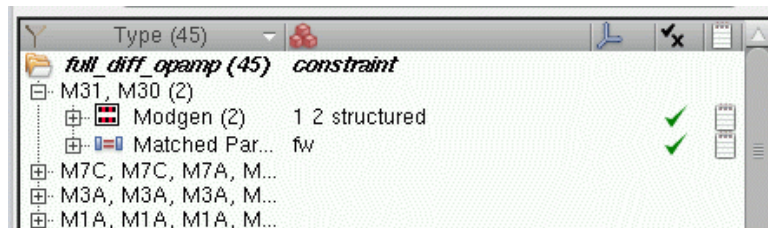


From here you can choose how you want to view information in the *Constraint Manager*.

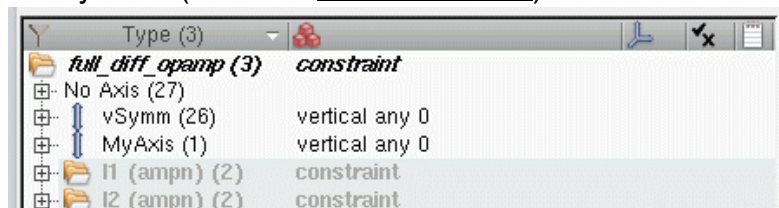
- *By Constraint:*



- *By Object Groups:*



- *By Axis (see also [Constraint Axis](#)):*



Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

- Selecting the *Sort By* sub-context-menu displays the following menu:



From here, you can choose the sort order category that you want to apply to constraint display in the *Constraint Browser*. For example, you can choose to sort the constraints by *Status*, which lets you to group together those constraints that currently have a *Passed* or a *Failed* status.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

- Selecting the *Show* sub-context-menu displays the following menu:

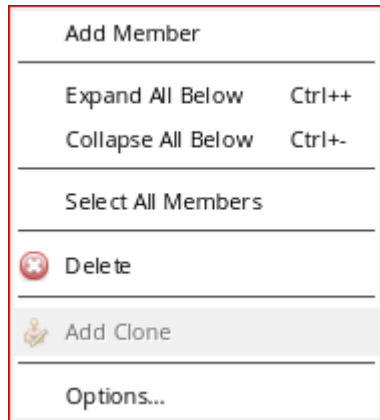


From here, you can choose (filter) what constraint information is to be displayed. This could be to filter to only display constraints that have been created during a certain period, or to only display constraints of a certain type.

Note: This works in the same manner as [Constraint Filters](#) and choosing *Options* from this menu will directly display the Customize Constraint Filters option.

Constraint Browser Table Context-Sensitive Menu

If you right-click the *Constraint browser* table area, the following context-sensitive menu is displayed:



Option

Add Member

Expand All Below

Description

Adds a constraint member to the selected constraint.

For more information see [Adding Constraint Members](#).

Expands the directories, templates, or hierarchy based on the way you accessed the context-sensitive menu to select the *Expand All Below* option.

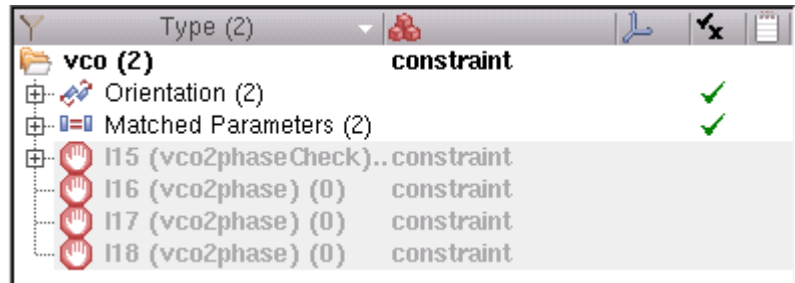
- When you right-click a constraint and select the *Expand All Below* option, the constraint is expanded and its members are visible.
- When you right-click a template and select the *Expand All Below* option, the template is expanded along with its constraints.
- When you right-click a directory and select the *Expand All Below* option, the entire hierarchy of the directory is expanded, be it a template or a constraint.
- When you right-click the root directory and select the *Expand All Below* option, the entire hierarchy is expanded, be it a template, constraint, or a directory.

Virtuoso Unified Custom Constraints User Guide

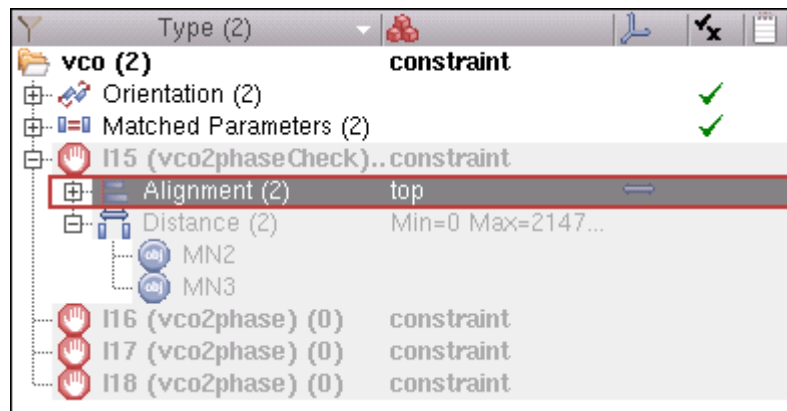
The Constraint Manager Assistant

Collapse All Below

Collapses all the directories available in the Constraint Manager window.



However, if a constraint is currently selected, then choosing this option will collapse the constraint to hide the members of the current constraint.



Note: This option does not apply when used against the selected constraint member.

Select All Members

If a constraint is currently selected, choosing this option will select all the member constraints in the browser (as shown in figure 1-10), in the *Navigator*, and on the design canvas.

Note: This option does not apply when used against the selected constraint member.

Delete

Deletes one or more selected constraints.

Add Clone

Creates one or more clones in the schematic with the same topology as the selected clone source. For steps, see [Adding Clone Def to an Existing Clone Family Template](#).

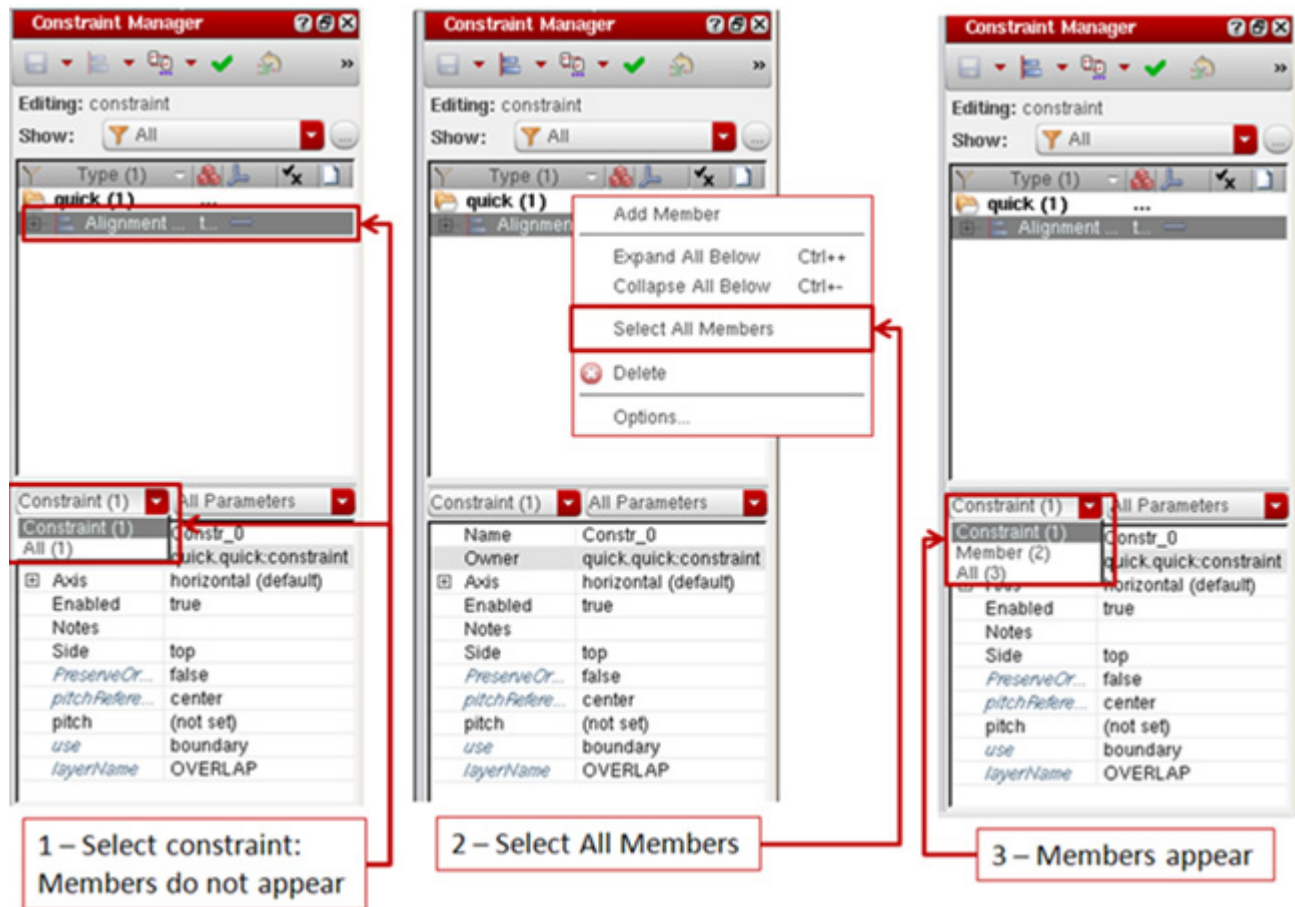
Options

Displays the [Constraint Manager Options Form \(Schematic\)](#).

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Figure 1-10 Using Select All Members Context-Sensitive Menu



Constraint Manager Options Form (Schematic)

Selecting *Options* from the Constraint Manager file menu displays the Constraint Manager Options form. The options form displayed in the schematic view is shown below. To view this same form in the layout editor, refer to the steps covered in the [Constraint Manager Options Form \(Layout\)](#) section.



Tip

Alternatively, you can select *Options* from the [Constraint Browser Table Context-Sensitive Menu](#) to display the Constraint Manager Options form.

The screenshot shows the 'Constraint Manager Options' dialog box with the following sections and controls:

- Global Constraint View List**
 - Text: "Your design does not have a configuration. To change your constraint views, edit the global view list."
 - Global View List:
- Halos**
 - ☒ Enable
 - Setup...** button
 - Halo Limit:
- Advanced Options**
 - ☐ Show Folders
 - ☐ Enable Constraint Overrides
 - ☐ Enable Move Constraints to/from Current Cell
- Selection Behavior**
 - ☒ Select constraints after creation
 - Selection Limit:
 - Cross-selection Limit:
- Buttons: OK, Cancel, Apply, Help

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Option	Description
<i>Global Constraint View List</i>	For more information on this option, see Changing a Constraint View Name List .
<i>Halos</i>	■ Checking the <i>Enable</i> option switches on the use of halos to highlight constraints in the design canvas. ■ Selecting the <i>Setup</i> option displays the Set Constraint Halo Options form, providing designers with the capability to set particular halo colors, and their style attributes, to one or more chosen constraints (see also Configuring Constraint Halo Display). You can set a <i>Halo Limit</i>, which is set to a maximum of 32 constraints by default. This is the maximum number of constraints that are haloed. Note: For more information on the use of halos, see Halo Highlighting.
<i>Advanced Options</i>	You can select the <i>Show Folders</i> check box to see the directories for sub-cell instances as well as the constraints at the current level schematic. These directories will be shown corresponding to all the children of the current cellview, initially with a question mark icon. Selecting the <i>Show Folders</i> check box activates the <i>Enable Constraint Overrides</i> and <i>Enable Move Constraints to/from Current Cell</i> check boxes. ■ Selecting the <i>Enable Constraint Overrides</i> check box adds a <i>Create Override</i> option to the Constraint Manager Constraint Browser Table Context-Sensitive Menu. For more information see Overriding Constraints. ■ Selecting the <i>Enable Move Constraints to/from Current Cell</i> check box adds <i>Move to Sub-Cell</i> and <i>Move to Current Cell</i> options to the <i>Constraint Manager</i> Constraint Browser Table Context-Sensitive Menu.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

For more information, see [Moving Constraints to Current or Sub-Cells](#).

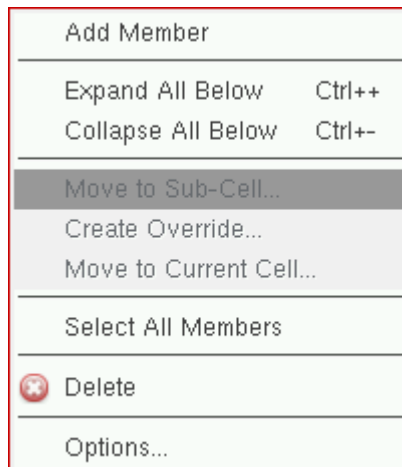


Figure 1-11 Constraint browser table context-menu with advanced options added

Selection Behavior

By default, the *Select constraints after creation* check box is selected in the *Selection Behavior* group box. As a result, after a constraint is created, it appears as selected in the *Constraint Browser* pane.

When a large number of constraints are created simultaneously, this process of selecting the created constraints automatically can affect performance adversely. To prevent performance issues, you can use the following options:

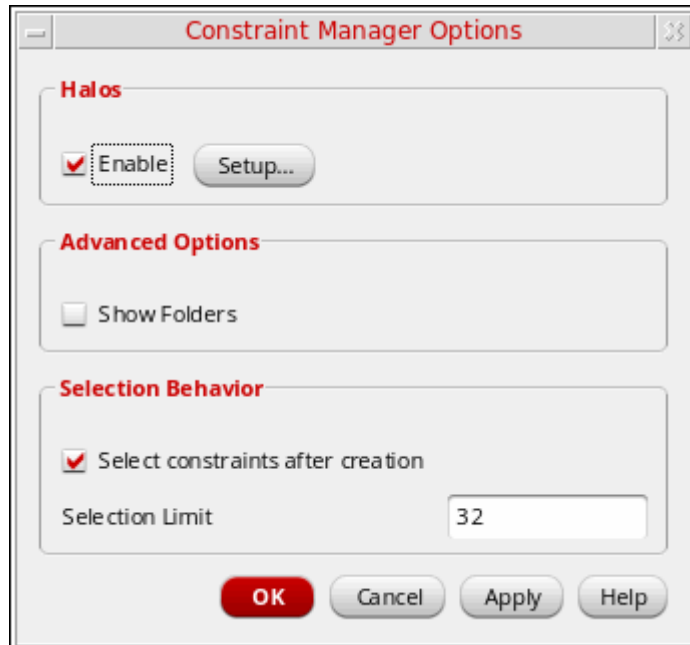
- *Selection Limit*: set to a maximum of 32 constraints by default. If the number of constraints created is greater than this limit, the constraints are not selected automatically after they are created.
- *Cross-selection Limit*: set to a maximum of 32 constraints by default. If the number of constraints created is greater than this limit, the constraints are not cross-selected automatically after they are created.



Setting a large selection and cross-selection limit is not recommended because it can slow down the constraint creation process, especially when the constraints also need to be haloed.

Constraint Manager Options Form (Layout)

Selecting *Options* from the Constraint Manager Constraint Browser Table Context-Sensitive Menu will display the Constraint Manager Options form.



Option

Halos

Description

- Checking the *Enable* option switches on the use of halos to highlight constraints in the design canvas.
- Selecting the *Setup* option displays the Set Constraint Halo Options form, providing designers with the capability to set particular halo colors, and their style attributes, to one or more chosen constraints (see also Configuring Constraint Halo Display).

Note: For more information on the use of halos see Halo Highlighting.

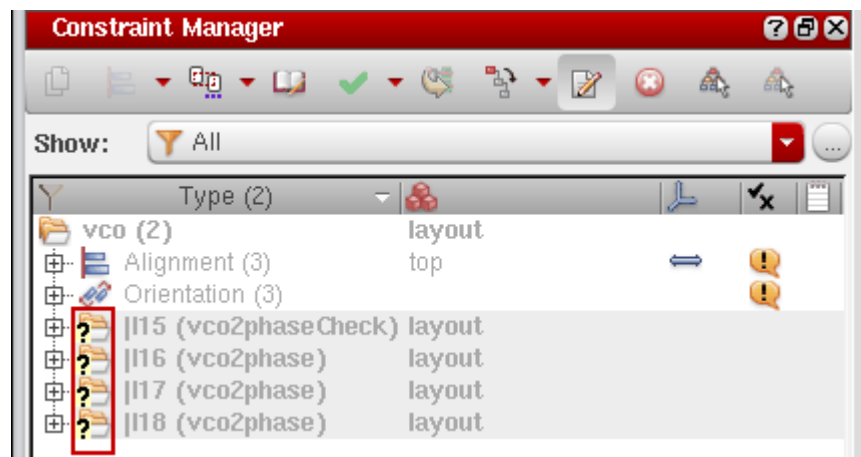
Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

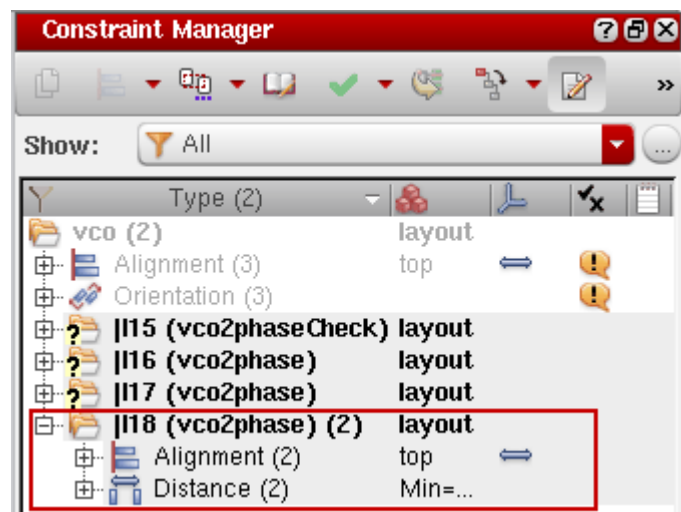
Advanced Options

You can select the *Show Folders* check box to see directories for sub-cell instances as well as constraints at current level layout. These directories will be shown corresponding to all children of the current cellview, initially with a question mark icon.

Note: The question mark icon indicates that the hierarchy inside the child has not been checked for the existence for any constraints.



On clicking the plus icon against a directory name, the number of constraints inside that directory will be displayed. In addition, the question mark will be removed against the directory icon in case there are constraints inside the directory.



Selection Behavior

By default, the *Select constraints after creation* check box is selected in the *Selection Behavior* group box. As a result, after a constraint is created, it appears as selected in the *Constraint Browser* pane. When a large number of constraints are created simultaneously, this process of selecting the created constraints automatically can affect performance adversely. To prevent performance issues, you can set a *Selection Limit*, which is set to a maximum of 32 constraints by default. If the number of constraints created is greater than this limit, the constraints are not selected automatically after they are created.



Setting a large selection limit is not recommended because it can slow down the constraint creation process, especially when the constraints also need to be haloed.

Soft Blocks

In the *Constraint Browser* of the layout window, directories for soft block instances are displayed right after the current level constraints. You are allowed to create pin and boundary constraints in the soft blocks in the L1 editing mode.

You can create boundary constraints on a soft block by selecting the soft block boundary. This can be done by making the soft block objects selectable in the object palette and using the constraint creation menu.

You can also create pin constraints on soft block pins from the current level. Constraints created on boundaries and pins of soft blocks are stored in the soft block itself and not at the current level. However, once constraints are created, they will be displayed inside directories for soft block instances.

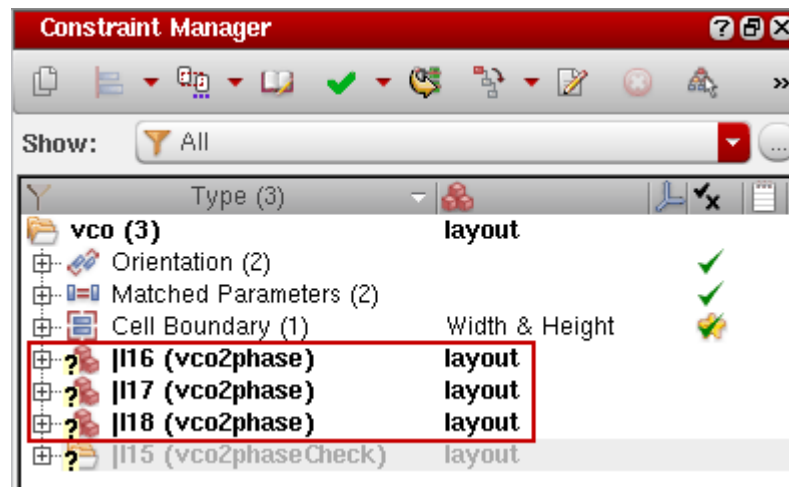
When the soft block instances are editable, you can also modify the constraint parameters for pin and boundary constraints on soft blocks. You can also delete the existing pin and boundary constraints for these soft blocks.

Unlike other directories, the soft block directories will not appear grayed out unless the soft block is in the read-only mode. In addition, if the soft block contains instances then the hierarchy below those instances will be shown but will remain non-editable.

Virtuoso Unified Custom Constraints User Guide

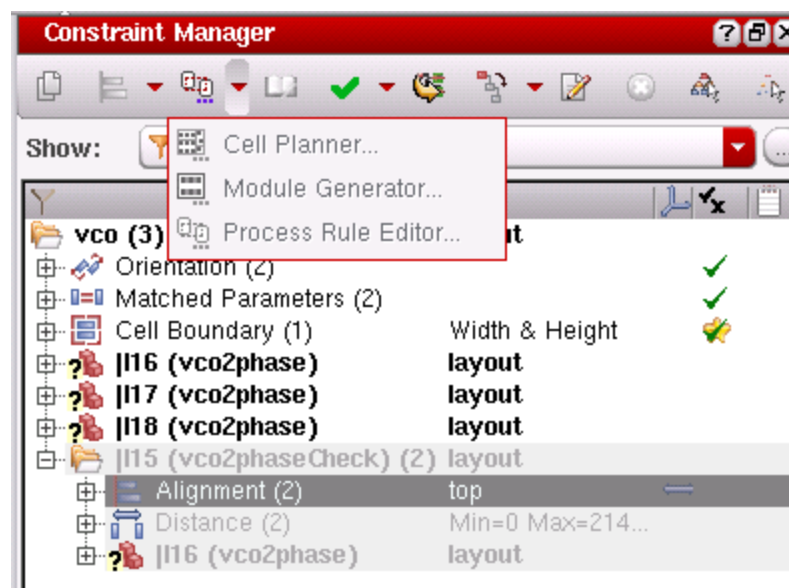
The Constraint Manager Assistant

The soft block icons are used to indicate soft block instances in the *Constraint Browser*, as highlighted in the figure below.



Finished Blocks

Constraints inside directories that represent instances for finished blocks will appear grayed out and remain non-editable. In addition, when a constraint inside a finished block or a soft block is selected, all editors — *Module Generator*, and *Process Rule Editor*—will be disabled.



Stop Point Icons

In earlier releases, the directories disappeared from the schematic *Constraint Browser* below the layout stop points whenever the Layout XL window is opened. Layout stop points are schematic symbols for which a layout master existed, as per the specified layout view list. However, now a logical hierarchy is displayed in the schematic *Constraint Browser* below stop points. The stop points are indicated visually using the stop point icons. Therefore, whenever you open the VLS XL window, the directory icons in VSE XL will automatically change to the stop points icons. When you close the VLS XL window, the stop-point icons will change back to directory icons.

Constraint Browser Columns



The *Constraint Manager* has the following sortable, and re-positionable, columns:

- *Type* (see [Constraint Types](#) and [Default Constraint Types](#))
- *Axis* (see [Constraint Axis](#))
- *Parameters* (for information on all the parameters associated with each default constraint type see [Default Constraint Types](#)).
- *Status* (see [Constraint Status](#))
- *Notes* (see [Constraint Notes](#))

Note: Right-clicking over the *Constraint Browser* column headers displays the [Constraint Browser Column Header Context-Menu](#).

Constraint Types

The *Type* column in the *Constraint Browser* displays the constraint type, for example *Symmetry*, *Alignment*, and *Orientation*.

Note: You can alphabetically sort on the constraints by clicking the *Type* column header.

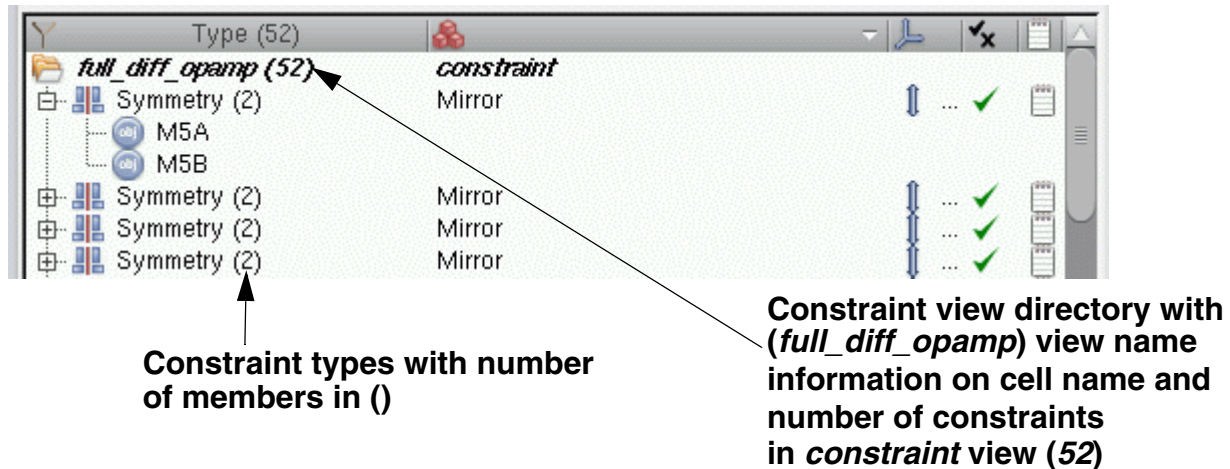
A constraint type is a formal definition of a design annotation that describes required electrical or physical relations between a set of design objects (constraint *members*) through a set of pre-defined constraint parameters, such as symmetry and orientation.

The number in parenthesis after each constraint type shows how many members (instances, nets, terminals, pins, and so on) that each constraint has. A directory is also displayed directly below the *Type* column header. This directory is [The Constraint View](#) directory for the current constraint view and provides information on the cell/view name and the number

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

of constraints in the current constraint view.



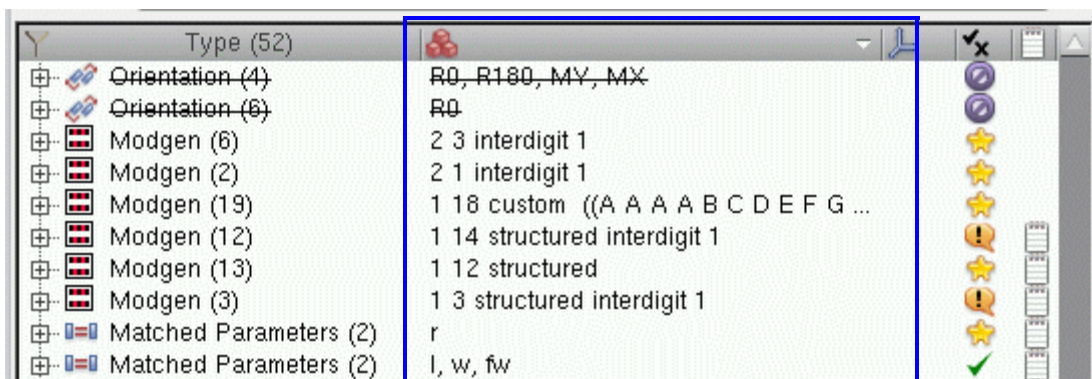
Each constraint type must be a member of one constraint category. Constraint categories allow you to manage the loading and viewing of constraint types. For example, when retrieving the constraint cache for a design, applications can specify the categories of constraint that they are interested in. Constraints retrieved from the cache can also be filtered according to category.

You can create new categories and also place constraints in additional categories to allow you to view your constraints in a more convenient way. For information on creating categories see [Customizing the Circuit Prospector](#) in the *Virtuoso Unified Custom Constraints Configuration Guide*.

- For information on default constraint types see [Appendix A, “Default Constraint Types”](#).

Constraint Parameters

The *Parameters* column in the *Constraint Browser* summarizes the important constraint parameter and value settings for each displayed constraint.



Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

A pre-defined constraint parameter describes the required electrical or physical implementation to the constraint-aware tools.

Constraint members will also often have their own parameters and values and these can be summarized when a constraint is expanded to reveal its members (see [Constraint Browser Features](#)).

Note: Parameters will, by default, inherit the parent constraint's value for that parameter. If the parent does not have such a value, the parameter value will be set to the default value for the member type.

Both constraints and their member parameter values can be edited using [Constraint Parameter Editor](#) in the lower section of the *Constraint Manager*.

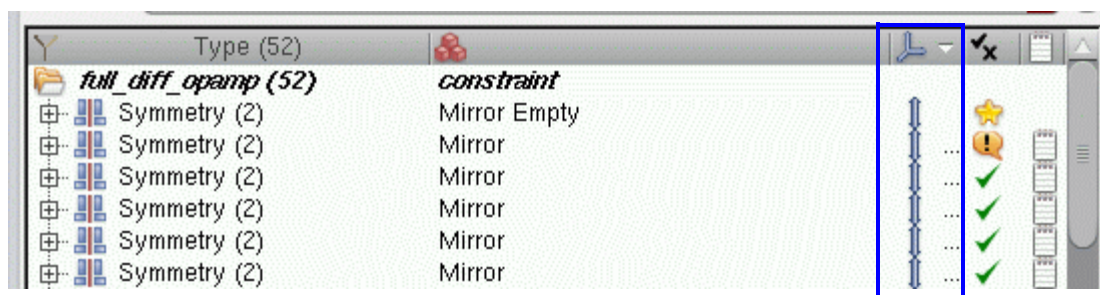
For information on all the parameters associated with each default constraint type see [Appendix A, "Default Constraint Types"](#).

Member Parameters

Individual constraint members can also have their own parameters. For example, each member of an *alignment* constraint can have a specific "*side*", such that in an *alignment* of 5, 3 of them have their top side aligned and the remaining 2 have their bottom side aligned. Only those parameters that make sense to be applied to members can be set on members.

Constraint Axis

The *Axis* column in the *Constraint Browser* displays information on axis direction, either vertical or horizontal.



A constraint axis is a virtual reference support object used by symmetry, alignment, and distance constraint types to define the direction and location of symmetrical pairs and aligned objects.

Note: *Symmetry*, *Distance* and *Alignment* constraint types should always reference an axis.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Axes then are constraint specific objects that are associated with particular Default Constraint Types (mentioned above). Each axis can be associated with more than one constraint, so that multiple symmetries and multiple alignments to the same axis can exist.

Constraints axes can only be created if an axis is created and stored in the constraint cellview in the front-end.

The following table describes the parameters associated with constraint axes.

Axis Parameter	Type	Default Value	Allowed Values
direction	enumerated list	vertical	vertical horizontal
axisLocation	boolean	any	any fixed
coordinate	integer	0.0	-
realizedCoordinate	integer	0.0	-

Symmetry and *Alignment* constraints do not have their own direction and coordinate parameters, but instead should be associated with an axis.

See also Editing the Constraint Axis Parameter on page 177.

Constraint Status

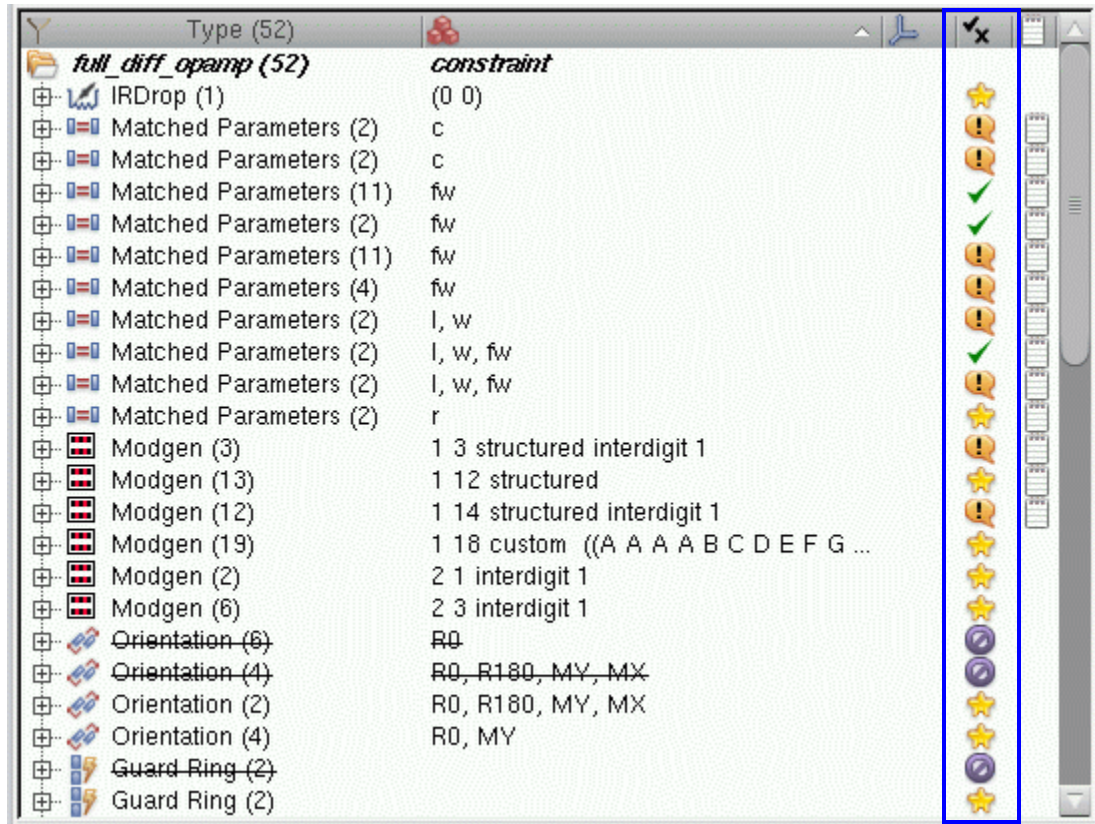
The *Check* option on Constraint Manager Toolbar can be used to check the enforcement status of constraints in the schematic or layout. Additionally, for example, the status of each layout constraint, with *respect to the schematic, can be updated dynamically, or on demand, using the *Update Schematic Constraints* command.

** For example, if it exists in the schematic or not, or if its parameters match its schematic counterpart.*

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

A constraint's current status is indicated using symbols and strikethrough text.



Note: You can click the *Status* column header to sort based on the constraint status.

The *Status* column also displays the status of constraint templates. For more information, refer to the [Viewing Template Status in Constraint Manager](#) section.

Constraint Status Icon Constraint Name and Definition

New



Informs that the constraint is new in the current view (for example, the schematic view), but does not yet have a corresponding constraint in the other view (for example, the layout view).

Different



Inform that there is a difference between this constraint and the corresponding constraint in the other view.

The related constraint status tooltip describes what the differences are, which could be any or all of the constraint parameters, member parameters, member order, or axis parameters.

Note: The Constraint Comparison Report can be used to examine any differences in more detail. For more information, see [Comparing Constraints between Schematic and Layout](#).

Passed



Inform that the constraint has been met successfully.

Inactive



Inform that the constraint is not currently active. This means that either it has `enabled=false` or it is out-of-context (that is, the constraint is in the same hierarchy, but as part of a different traversal), or it has been overridden. Again, the related tooltip provides more details.

Failed



Inform that the constraint has not been met (validated).

For example, depending on the constraint, it is possible the parameters do not match, are not aligned, are not symmetrical, and so on.

New and Passed



Inform that the constraint is new in this view, and has also been marked as passed.

New and Failed



Inform that the constraint only exists in the current view and that it has also failed in this view.

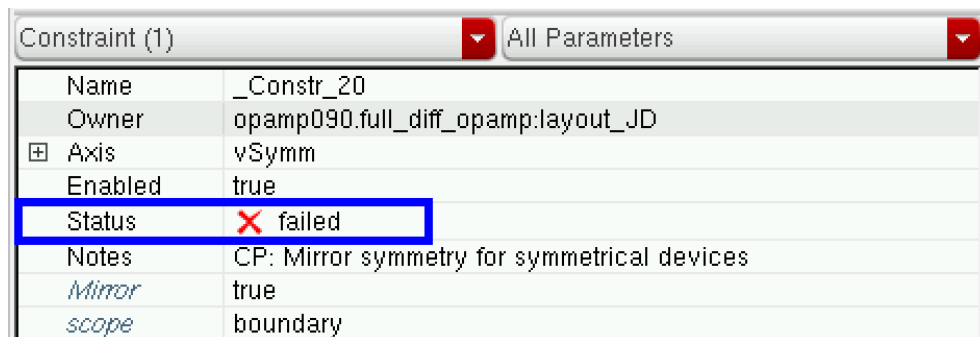
(No Icon)

A constraint that is blank, with no icon, represents a constraint that has not yet been checked.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Note: In the layout view, the latest constraint *Status* information is also displayed in the *Constraint Parameter Editor* pane of the Constraint Manager assistant, as shown below.



Constraint (1)		All Parameters
Name	_Constr_20	
Owner	opamp090.full_diff_opamp:layout_JD	
Axis	vSymm	
Enabled	true	
Status	✖ failed	
Notes	CP: Mirror symmetry for symmetrical devices	
Mirror	true	
scope	boundary	

Filtering Constraint Status

You can use [Constraint Filters](#) to only show in the *Constraint Browser* those constraints that have, for example, *Passed* or *Failed* status.

Constraint Conflicts

A conflict can be something that causes a constraint to fail its consistency check. For example, an instance cannot be a member of more than one *symmetry* constraint. If a *symmetry* constraint with members `/I2/MP0` and `/I2/MP1` is transferred to layout where the members are mapped to `|I2|MP0` and `|I2|MP1`. Creating a new *symmetry* constraint in layout with members `|I2|MP0` and `|I2|MP2` would override the transferred constraint because the `|I2|MP0` member causes a conflict.

For more information, see [Constraint Transfer](#) in the Virtuoso Layout Suite documentation.

Note: There is no verification check to ensure that a constraint set by a schematic is achievable in layout. Individual applications are responsible for marking constraints as *passed* incrementally as they are run.

Constraint Notes

In the *Constraint Browser*, if a page icon is displayed in the *Notes* column, it indicates that a note has been associated with the corresponding constraint.

Designers can add notes to constraints to provide more details about the purpose of the constraint or to record and special considerations for implementation.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Constraint notes cannot be added to constraint members. However, any existing constraint notes will be transferred to a new constraint when a constraint is moved to the current cell (see [Moving a Constraint to the Current Cell](#)) or to a sub-cell (see [Moving a Constraint to Sub-Cells](#)). In the case of moving to a sub-cell, all constraints will show the same note.



Figure 1-12 Notes have been added to a number of symmetry constraints

Any notes that have been applied to a constraint can be accessed from the *Notes* column in the *Constraint Manager*. You can see the content of a note by moving the mouse cursor over the *Note* icon to raise a tooltip.

By default, the note's tooltip displays maximum 30 characters in a line width and 10 lines in height. If a line in the notes contains more than 30 characters, the characters after the 30th one get replaced with an ellipsis symbol in the tooltip. Similarly, if a note has more than 10 lines, the lines after the default 10th line are replaced with an ellipsis symbol in the tooltip.

To control the number of maximum characters in a line width or maximum number of lines that are visible in any constraint note's tooltip, use the [noteTooltipMaxCharWidth](#) and [noteTooltipMaxLines](#) environment variables.

For information about editing the tooltip content, see [Editing the Note Parameter](#).

Creating Constraints in the Constraint Manager

Constraints can be created at the current level in the design hierarchy with *members* specified at different levels as you descend and return through a hierarchy, or by selecting the objects from any level in the *Navigator Assistant* (for an example of use, see [Creating a Shielding Constraint and Setting a Shield Net](#)).

Note: See also [Creating New Constraints](#) in the *Virtuoso Unified Custom Constraints Configuration Guide*.

When creating a constraint at the top level of a design, the name of the cell will be displayed in the *Constraint Browser* with the constraint view name and the number of constraints shown in separate brackets after it.

Constraint Creation Basics

This section covers nuances that you need to understand before starting to create a constraint.

Constraint Members

Constraint members can be instances, nets, pins, or terminals depending on the type of constraint.

Note: For information on what type of members a particular constraint can have see [Default Constraint Types](#).

A constraint member is specified by a fully qualified path from the root of the hierarchy, for example:

```
"(amsPLL.vco:schematic) /I18 (amsPLL.vco2phase:schematic) /MN0 (gpdk.nmos:symbol)"
```

Constraint Creation and Current Object Selection

A particular constraint can only be created if the correct *number* of a specific object *type* is currently selected in the design canvas (or in the *Navigator* assistant). For example, a *Matched Length* constraint can only be created if two or more nets are currently selected.

If you attempt to create a constraint that only requires (max) one member, but have multiple candidate members currently selected, then the equivalent number of constraints of that type will be created as a result of your current object selection. For example, a *Process Rule Override* (PRO) constraint requires that a single net, net class, or design be selected to create a single PRO constraint. Therefore, if you have three nets currently selected, for

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

example, this would create three PRO constraints if you then chose to create a constraint of that type.

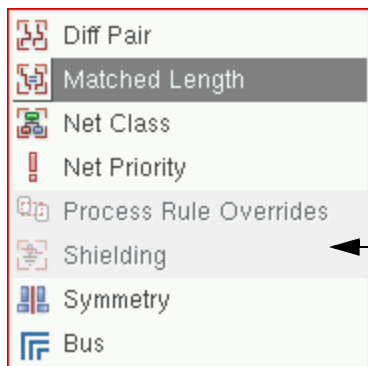
If you have a range of object types selected, all constraints that can be created from the current object selection set will be shown as available. Those objects that are not required, or allowed, for a particular constraint will then be ignored on the creation of that constraint. For example, if you have two nets and two pins currently selected, the two pins will be ignored when creating a *Matched Length* constraint. This therefore allows you to do an area select on the canvas, rather than individually selecting the exact object types that you want to be constraint members.

Some constraints require all objects to be of the same type. This can therefore lead to a situation where there are sufficient objects currently selected to allow for the creation of two different constraints. For example, if you have two instances and two nets currently selected and then choose to create a *Symmetry* constraint. The result will be that a *Symmetry* constraint will be created for the selected instances and another, separate constraint, for the selected nets. If one of these constraints is not required, it can be deleted (see [Deleting Constraints](#)).

To aid constraint creation, those constraints that cannot be created, due to the current object selection set, will be grayed (filtered) out in the *Constraint Creation* menu on [Constraint Manager Toolbar](#). Those constraints that can be created will be shown as active/selectable.

Note: Constraints are categorized by type, as either *Electrical*, *Placement*, *Routing*, or *Rapid Analog Prototype* in the *Constraint Creation* menu.

For example, if you have (only) two nets selected in the design canvas, the types of *Routing* constraints that you can create on them appear as enabled in the sub-menu and others are disabled, that is, grayed out, as shown below.



Process Rule Overrides and Shielding constraints cannot currently be created when 2 nets are selected.

Note: The list of available constraints will dynamically update depending upon the currently selected object set.

Steps to Create Constraints in the Constraint Manager

To create a new constraint using the *Constraint Manager*:

1. If not already on display, access the *Constraint Manager*.

For information on the various methods of accessing the *Constraint Manager*, see [Accessing the Constraint Manager](#).

Note: When the *Constraint Manager* is invoked it will take account of any current object selections in the design canvas and *Navigator*.

2. Optionally, descend to the level of the design hierarchy that you want to create the constraint in.

You can descend the design hierarchy using *Edit – Hierarchy – Descend Edit*.

For more information see [Creating Constraints at Different Design Levels](#).

3. (If not already selected) In the current design session window, or in the *Navigator Assistant*, select those design objects (instances, nets, pins, or terminals) that you want to include as constraint members in the new constraint.

Note: You can create constraints that comprise of member objects located at different levels of the hierarchy by using the cell view tabs on display. Ensure that the `shift` key is selected when you are clicking the objects to be constrained. Again, selection can be made on the canvas or using the *Navigator*.

Some constraint types have lower and upper limits to the number of members that they comprise of, and also the type of object members that can be included in a constraint. For example, it might not be possible to mix instances and nets in the same constraint. If an anomaly is identified, a warning message is displayed informing you that you will not be able to create the constraint.

Note: For information on what object members (and their number) that can be included in a particular constraint see [Default Constraint Types](#).

For certain constraints the *order* in which you select member objects can also be important. You can change member order using the *Constraint Manager* assistant (see [Display Constraint Type Members](#)). For more information on what constraints require correct member order selection again, see [Default Constraint Types](#).

4. Choose the type of constraint that is to be created by selecting it from the *Electrical*, *Placement*, *Routing*, and *Rapid Analog Prototype* sub-menus available within the *Constraint Creation* menu on the *Constraint Manager* toolbar.

Note: Only those constraints that can be created for the current selected object set will be shown as selectable. Those constraints that cannot currently be created will be

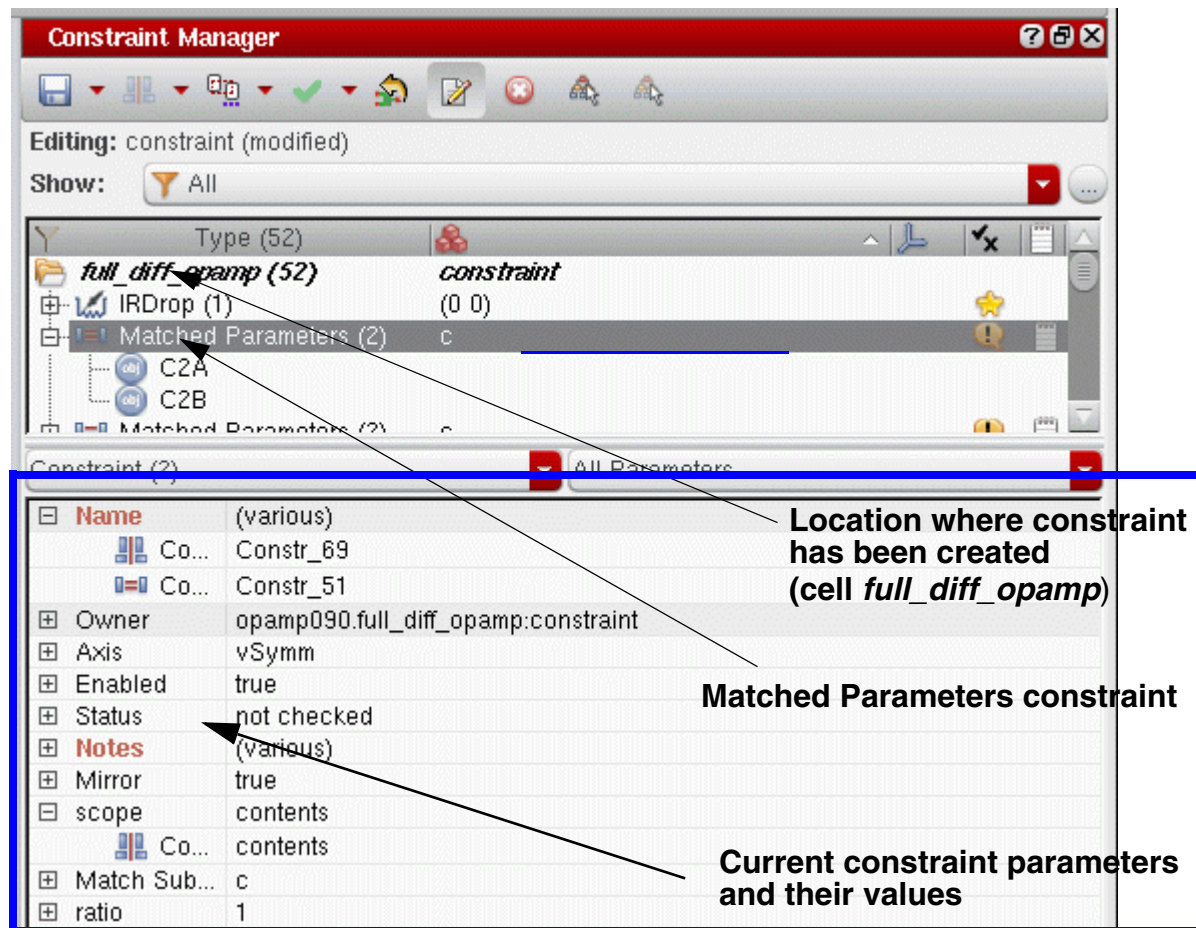
Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

grayed (filtered) out. For more information, see [Constraint Creation and Current Object Selection](#).

Assuming that constraint membership number and type rules are adhered to, the new constraint will be added to the *Constraint Browser* with a default constraint name applied, for example `Constr_69`.

Note: If you are creating a constraint for one or more iterated objects see [Creating Constraints for Iterated Objects](#).



Important

If [The Circuit Prospector Assistant](#) is currently open, and you have search results selected, choosing to create a new constraint will create one or more constraints based on the selected search results in the *Circuit Prospector* rather than the current selections in the design canvas or the *Navigator*.

Related topics

- For information on creating constraints using the *Circuit Prospector*, see [The Circuit Prospector Assistant](#).
- For more detailed information on creating constraints at different levels of the design hierarchy see [Creating Constraints at Different Design Levels](#).

Creating Constraints Using the BitSelector

While creating constraints, if you select the full iterated name in the `BitSelector`, for example `MN1<0:2>`, then one or more constraints will be created between the individual vector members (that is, `MN1<0>`, `MN1<1>`, and `MN1<2>`). The number of constraints that are created depends on the type of constraint chosen.

For most constraint types a single constraint will be created containing the individual vector names (for example, `alignment(MN1<0>, MN1<1>, MN1<2>)`). For symmetry and distance constraint types, multiple constraints will be created (for example, `symmetry(MN1<0>, MN1<2>)` and `symmetry(MN1<1>)`).

Note: If you select individual vector members from the `BitSelector`, one or more constraints of the selected type will be created dependent upon the number of vector members selected and the chosen constraint type.

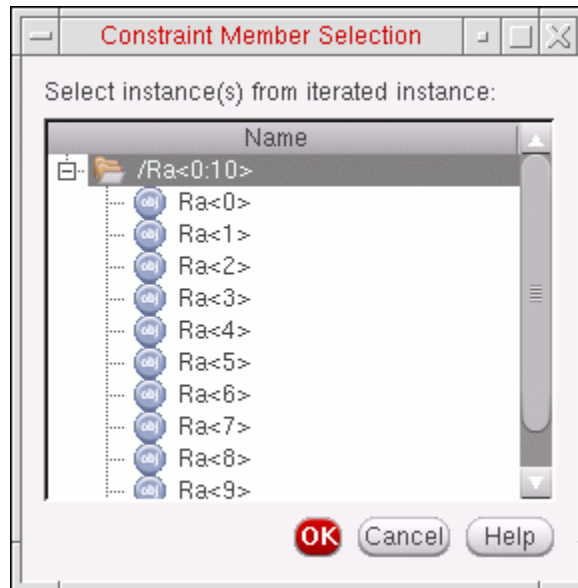
Creating Constraints for Iterated Objects

If you select one or more iterated instances, nets, pins, or `instTerms` from the design canvas or the [Navigator Assistant](#), the Constraint Member Selection form is displayed after you select the type of constraint that you want to create. A separate form, using a tree format, is displayed for each selected iterated instance.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Note: The Constraint Member Selection form is also opened when constraints need to be created between multiple selected busses and net bundles.



To create constraints for multiple iterated objects:

1. From the design canvas, or the *Navigator* assistant, select the iterated objects that you want to create a constraint between.
2. Choose the constraint type to be created by either:
 - a. Selecting the constraint type from the *Constraint Creation* menu on the *Constraint Manager* toolbar, or
 - b. Right-clicking over the *Constraint Browser* and selecting *Add Constraint*. From here, you can select the constraint type required.

The Constraint Member Selection form is displayed.

3. Select the iterated object *Name* that you want to create a constraint for and click *OK*.
4. Step 3 will be repeated until selections have been made for all iterated objects currently selected.

Once you have made all your selections for each selected iterated object the constraint will be created and displayed in the *Constraint Browser*.

Note: In Schematic, you can choose the *All Selected* filter in the Constraint Manager to show constraints on all the members of the iterated instance in the design canvas or the navigator.

Creating Modgen Constraints

Modgen constraints can be created *interactively* on selected instances in Schematic (or Layout) by doing one of the following:

- Choose the *Modgen* constraint generator from the list of constraint types in the *Placement* category. Then, run the Modgen Editor from the *Constraint Editor* menu to specify the pattern and other attributes.
- Run the Modgen Editor directly from the *Constraint Editor* menu.
- Launch any of the applicable constraint generators from the *Rapid Analog Prototype* category to create a modgen constraint as a template whose parameters are directly editable, that is, without launching the Modgen Editor. Such modgen constraints are called generic modgen constraints. For example, *GenericModgen* constraint generator can be used to create a modgen on any group of devices or structures as explained in detail in the [Creating Generic Modgen Constraints](#) section below.

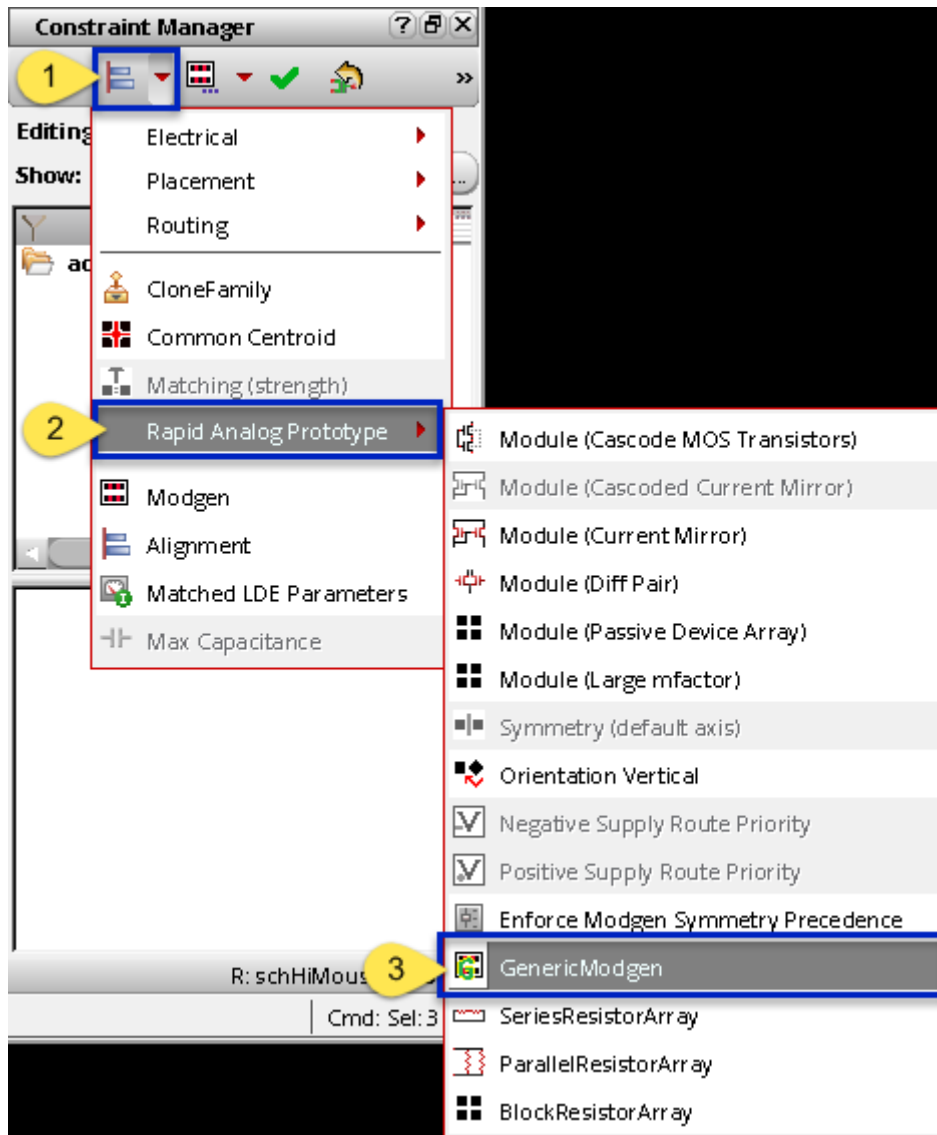
Creating Generic Modgen Constraints

The *GenericModgen* constraint generator enables you to create generic modgen constraints on one or more selected devices.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Run this constraint generator from the *Rapid Analog Prototype* sub-menu in the *Constraint Creation* menu of the Constraint Manager assistant.



Note: To enable the *GenericModgen* constraint generator option on the *Rapid Analog Prototype* sub-menu, ensure you have at least one device selected on the schematic or layout canvas.

Use this form to specify the device interdigitation pattern, orientation, default orientation, spacings, and abutment. The device interdigitation pattern can contain dummies and there are options for setting up the parameters on those dummies. You also have the options for adding guard rings and setting the modgen merge layer. The sections below explain use of each tab in detail and finally how to generate the generic modgen template.

Setting Patterns, Orientations, Dummies, Spacings, and Abutment

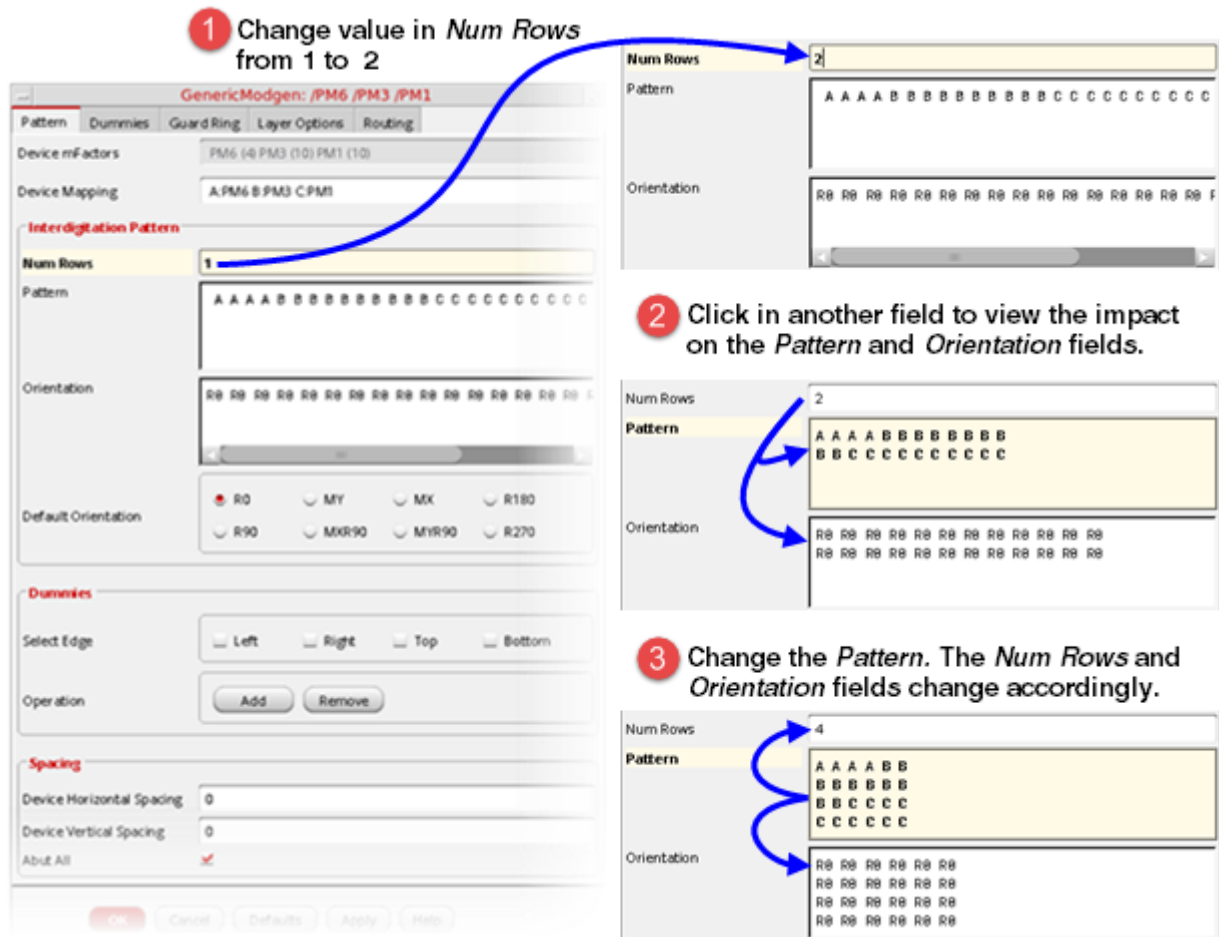
When you access the GenericModgen form, the *Pattern* tab is displayed by default. The first field, *Device mFactors*, is uneditable. It displays the mFactor information of the selected device. Use this tab as following to set the device interdigitation pattern, orientations, dummies, spacings, and abutment parameters:

1. In the *Device Mapping* field, specify the symbols to be used for creating the device interdigitation pattern. By default, the field shows the device mapping of the currently selected devices. If you edit this field, ensure that the symbol specification starts from A and continues in descending alphabetic order without skipping any alphabet. For example, A : PM6 B : PM3 C : PM1 this is valid; however, A : PM6 C : PM3 D : PM4 is invalid.
2. Specify the number of modgen rows in the *Num Rows* field within the *Interdigitation Pattern* group box. This number controls the pattern and orientation arrangement of the device symbols.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

The *Num Rows* field is interlinked with the *Pattern* and *Orientation* fields. When you change any one of these three fields, it results in corresponding adjustments in the other two field values. The following figure illustrates this:



Note: You can undo the changes made using the `Ctrl+Z` shortcut key, which works only in these three fields.

- Specify the device interdigitation *Pattern*. In this field, each device symbol must appear as many times as the mFactor of the corresponding device. The dummies are specified by an asterisk (*) and gaps by a hyphen (-).

Pattern	* A A B B *	Pattern	A2 B B C
	* C C D D *		C D D -

To add dummies on the left, right, top, and/or bottom of a device interdigitation pattern, you can alternatively use the options provided in the *Dummies* group box. This is an automated way of adding dummies around a pattern. However, if you need to add

dummies somewhere in between, you should use the manual mechanism of inserting asterisks in the *Pattern* field.

Note: The field supports the use of the modgen pattern editor shorthand notations. For example, A2 or (A) 2 is equivalent to A A.

4. Specify the device *Orientation*. Each orientation must appear as many times as the mFactor of the corresponding device. The orientation must be one of the following: R0, R90, R180, R270, MX, MY, MYR90, or MXR90.
5. Select the *Default Orientation* from the given radio buttons: R0, MX, MY, R180, R90, MXR90, MYR90, or R270. These radio buttons are useful for defining the default orientation of the dummies if any are added using the *Dummies* group box.

Note: In the *Default Orientation* section, the text adjacent to each radio button can be replaced with an icon of your own choice. For more information, refer to the [Replacing Text with Icons on the Pattern Tab](#) section.

6. Select the required check box adjacent to the *Select Edge* field in the *Dummies* group box. This specifies the side on which the dummies should be added in the pattern, that is, *Left*, *Right*, *Top*, and *Bottom*.
7. Click the required button, *Add* or *Remove*, to specify the *Operation* to be performed to add new or remove existing dummies.

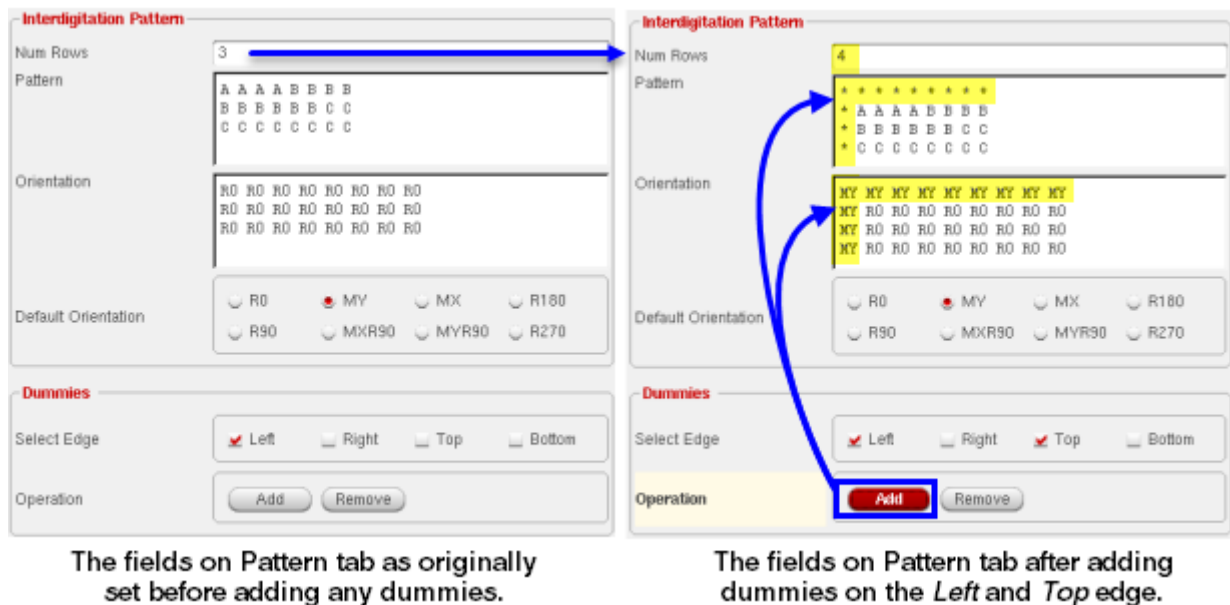
Note: In the *Operation* section, the text displayed on each button can be replaced with an icon of your own choice. For more information, refer to the [Replacing Text with Icons on the Pattern Tab](#) section.

If you click the *Add* button, on the selected edges, dummies (represented by asterisks) get added in the *Pattern* field and the *Orientation* field gets updated with an equal number of default orientation notations. For example, in the figure below, if you select MY as the *Default Orientation* and select *Left* and *Top* edges, the *Pattern* and

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Orientation fields start showing the added dummies. Also note the change in the *Num Rows* field when dummies were added on the *Top* edge.



Note: When you click *Add*, at a time, only one row of dummies is added on the specified edge. To add more than one row of dummies, click *Add* as many times as the number of rows you want. You can also add dummies of different default orientations (refer to the image below). To do this, ensure that you change the selected *Default Orientation*

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

before clicking *Add*.

Interdigitation Pattern

Num Rows

7

Updated when each row of dummy is added.

Pattern

```

* * * * *
* * * * *
* * * * *
* * * * *
* * A A A B B B B
* * B B B B B C C
* * C C C C C C C

```

Orientation

```

R90 R90 R90 R90 R90 R90 R90 R90 R90 R90 R90
R90 R90 R90 R90 R90 R90 R90 R90 R90 R90 R90
MY MY MY MY MY MY MY MY MY MY MY
MY MX MX MX MX MX MX MX MX MX MX
MY MX MX R0 R0 R0 R0 R0 R0 R0 R0
MY MX MX R0 R0 R0 R0 R0 R0 R0 R0
MY MX MX R0 R0 R0 R0 R0 R0 R0 R0

```

Default Orientation

☐ R0
☐ MY
☐ MX
☐ R180
☒ R90
☐ MXR90
☐ MYR90
☐ R270

Dummies

Select Edge

☐ Left
☐ Right
☒ Top
☐ Bottom

Operation

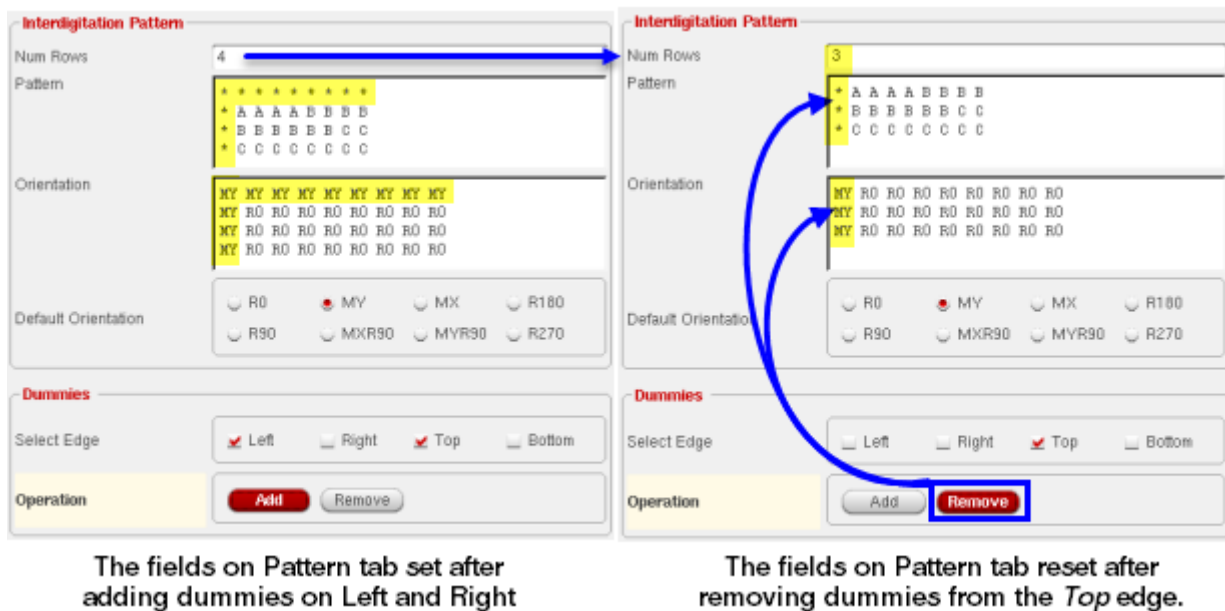
Add

Remove

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

To remove all dummies on the specified side, select the desired edge and click the *Remove* button (as shown in the image below).



Note: While removing the dummies, the *Default Orientation* selection is not considered as done while adding dummies.

8. Enter the *Device Horizontal Spacing* in the *Spacing* group box.

Note: The specified horizontal spacing has no effect if the *Abut All* check box is selected.

9. Enter the *Device Vertical Spacing*.

10. Select the *Abut All* check box if abutment of devices is needed.

Replacing Text with Icons on the Pattern Tab

On the *Pattern* tab of the GenericModgen form, you can replace the text with icons for the following:

- *Default Orientation* radio buttons displayed in the *Interdigitation Pattern* group box
- *Operation* buttons displayed in the *Dummies* group box

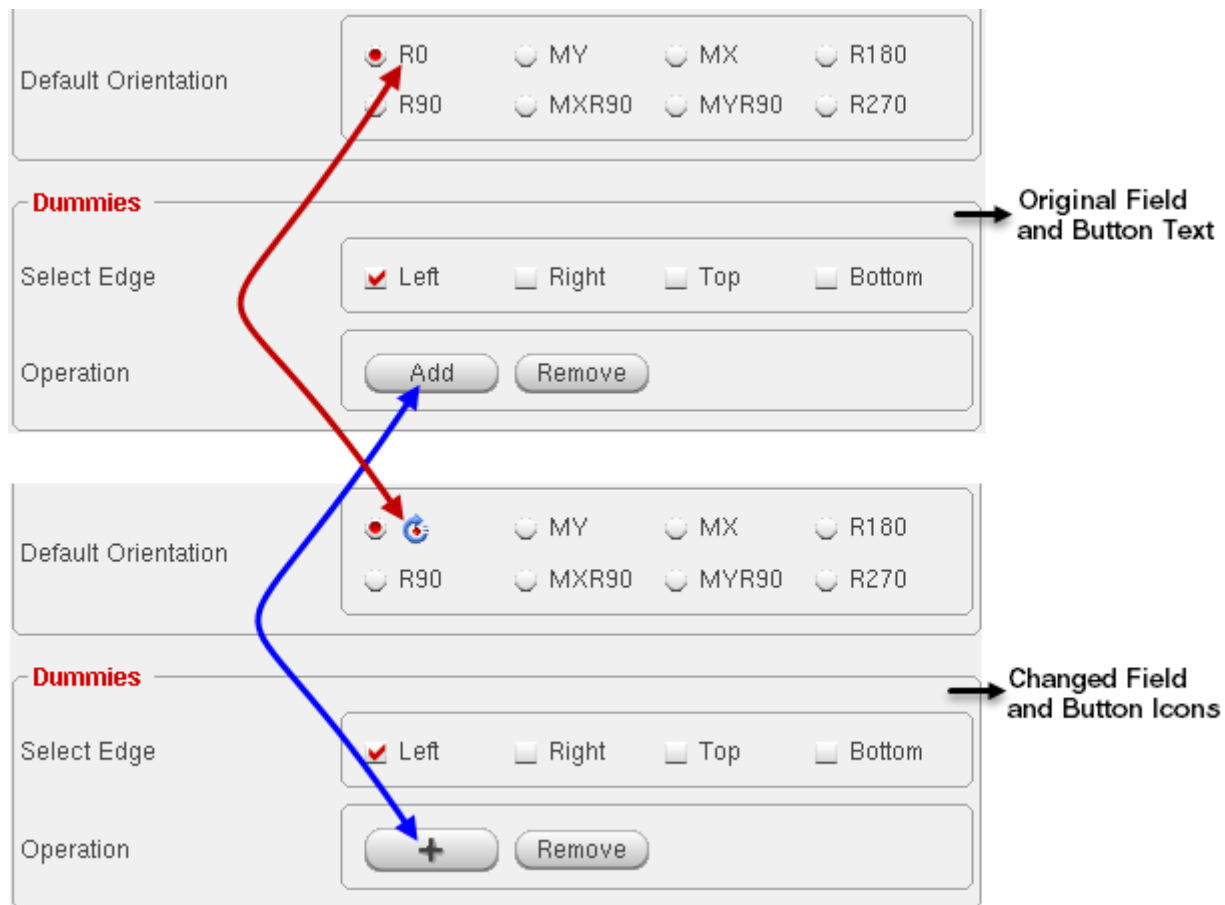
For this, create a `.cadence/icons/16x16` directory in your current working directory and save in it the required icon files with a `.png` extension. The name of the icon file should be the same as the text displayed on the form. For example, to replace the text of the *R0* radio

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

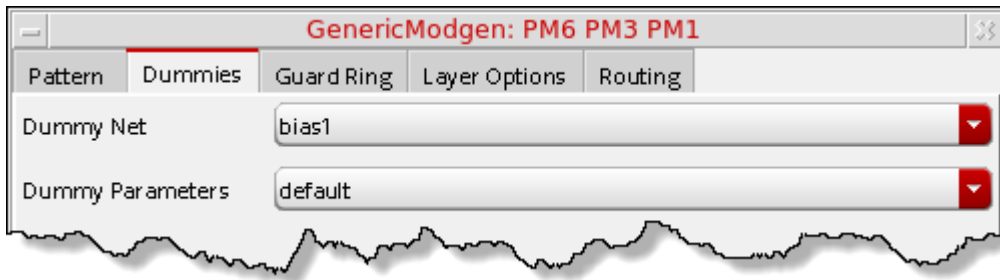
button, save the new icon file with the name `R0.png`. Similarly, to replace the text of the *Add* button, name the icon file as `Add.png`.

Note: If the GenericModgen form is currently open, the changed icons show on the form only after you close the form and reopen it.



Specifying Dummy Net

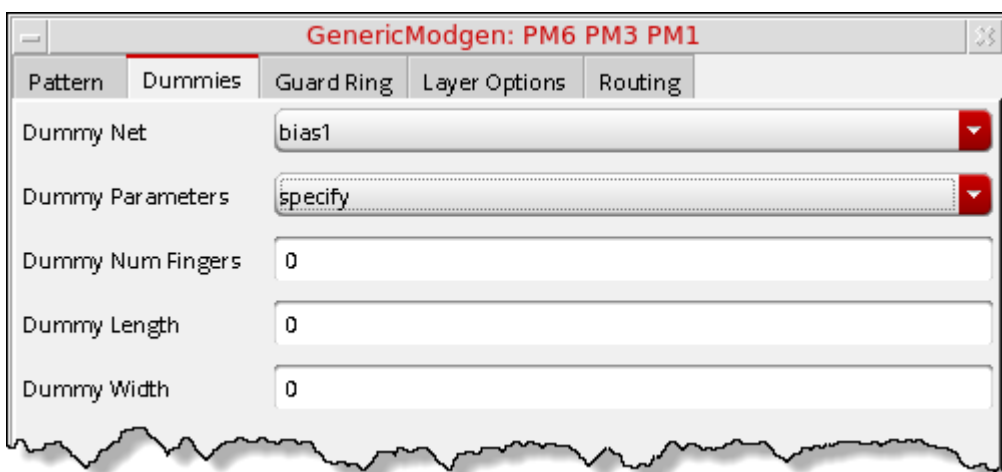
The *Dummies* tab has options for specifying the dummy nets. These options are the same as those available in the Modgen Editor.



The following steps explain use of each option available on the *Dummies* tab:

1. Select the required *Dummy Net* from the drop-down list box.
2. Select the option to determine from where the *Dummy Parameters* should be obtained. You have the following three options in the drop-down list box: *neighbor*, *default*, or *specify*.

If you choose *specify* from the *Dummy Parameters* drop-down list box, the following three additional fields are displayed to you: *Dummy Num Fingers*, *Dummy Length*, and *Dummy Width*.

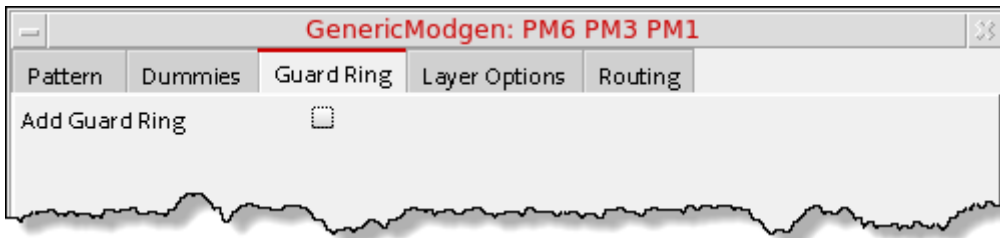


3. Add values of your choice to the additional fields displayed to specify the custom dummy parameters.

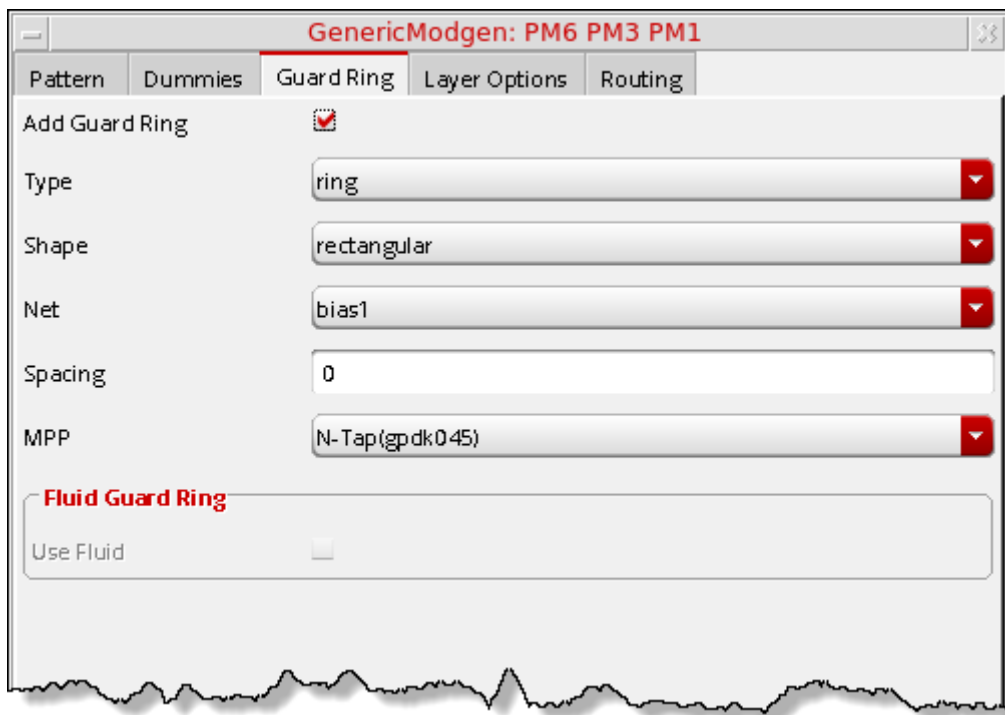
Note: Values previously specified in the additional fields are ignored if you later choose *neighbor* or *default* from *Dummy Parameters*.

Adding a Guard Ring Around Generated Modgen

The *Guard Ring* tab has options for adding a guard ring around a generated modgen. These options are the same as those available in the Modgen Editor. When you open this tab, you see only the *Add Guard Ring* check box.



On selecting this check box, the following fields also become visible: *Type*, *Shape*, *Net*, *Spacing*, and *MPP*.



Note: The *Use Fluid* check box in the *Fluid Guard Ring* group box appears as enabled only if your design supports creation of fluid guard ring devices.

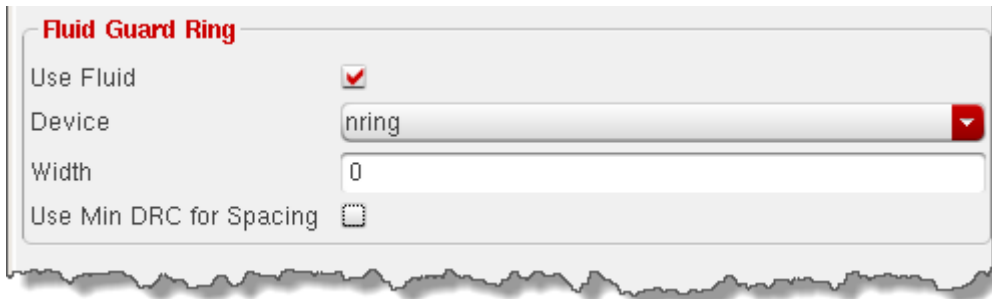
The following steps explain the use of each option available on the *Guard Ring* tab:

1. Select the *Add Guard Ring* check box to add the guard ring.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

2. Select the *Type* of guard ring. You have two options in the drop-down list box to choose from: *ring* or *pane*.
3. Select the *Shape* of the guard ring. You have two options in the drop-down list box to choose from: *rectangular* or *rectilinear*.
4. Enter a value in the *Spacing* field. This value determines the spacing between the modgen devices and the guard ring. This is in addition to the minimum DRC spacing rule.
5. Select the *MPP* to use for the guard ring.
6. If you want to create a fluid guard ring instead of a normal guard ring, select the *Use Fluid* check box. In this case, the value specified in the *MPP* drop-down list box becomes inapplicable and therefore, the list box disappears from the form, as shown below.

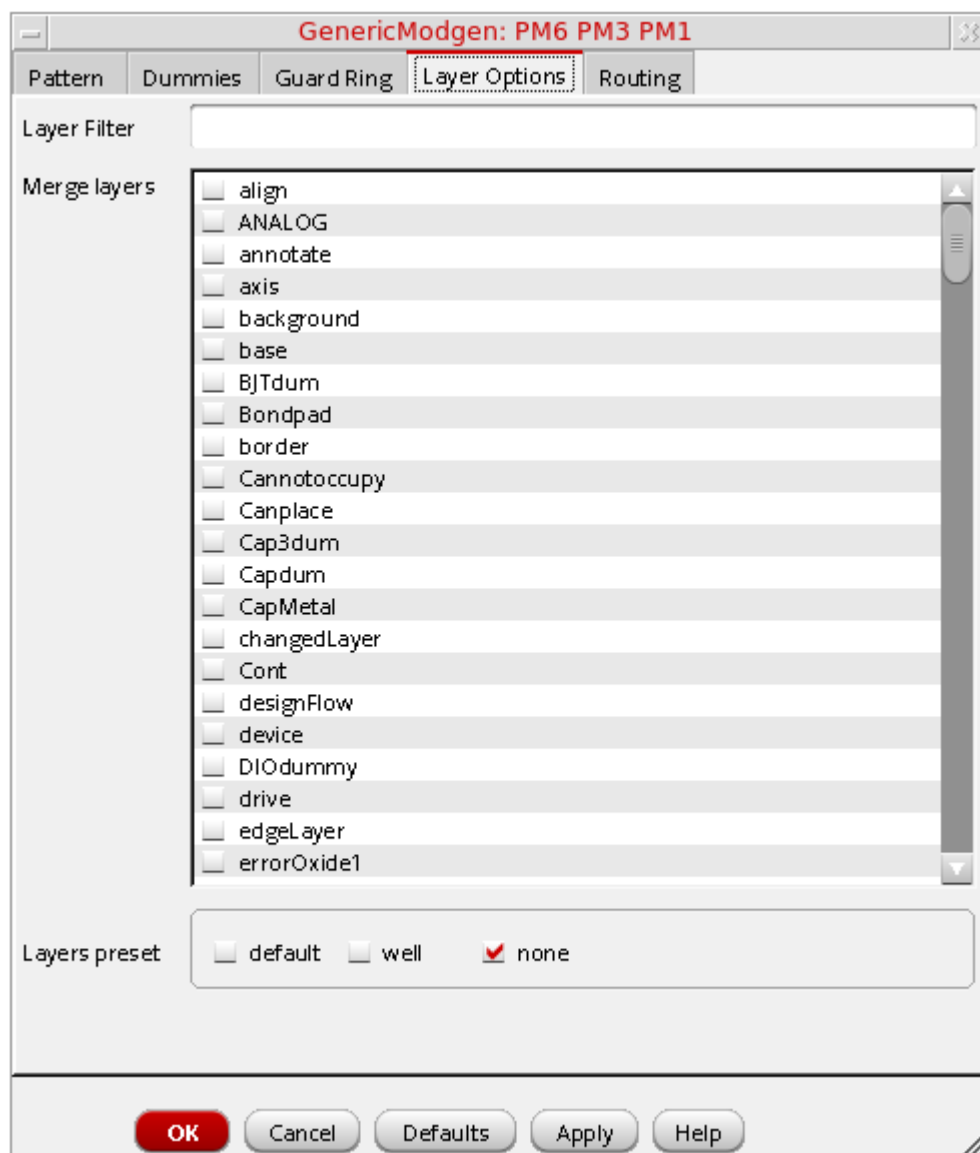


The screenshot shows a configuration panel titled "Fluid Guard Ring" in red text. It contains four fields: "Use Fluid" with a checked checkbox, "Device" with a dropdown menu showing "nring", "Width" with a text input field containing "0", and "Use Min DRC for Spacing" with an unchecked checkbox. The panel has a light gray background and a subtle shadow.

7. Select the *Device* around which the fluid guard ring needs to be created. If no device is specified in the technology file, this list displays the value, *undefined*.
8. Specify the *Width* of the fluid guard ring.
9. Select the *Use Min DRC for Spacing* check box if you want to create the fluid guard ring at the minimum DRC from the devices. When you select this check box, value specified in the *Spacing* field will not be used and therefore, it disappears from the tab.

Specifying Merge Layers for Generated Modgen

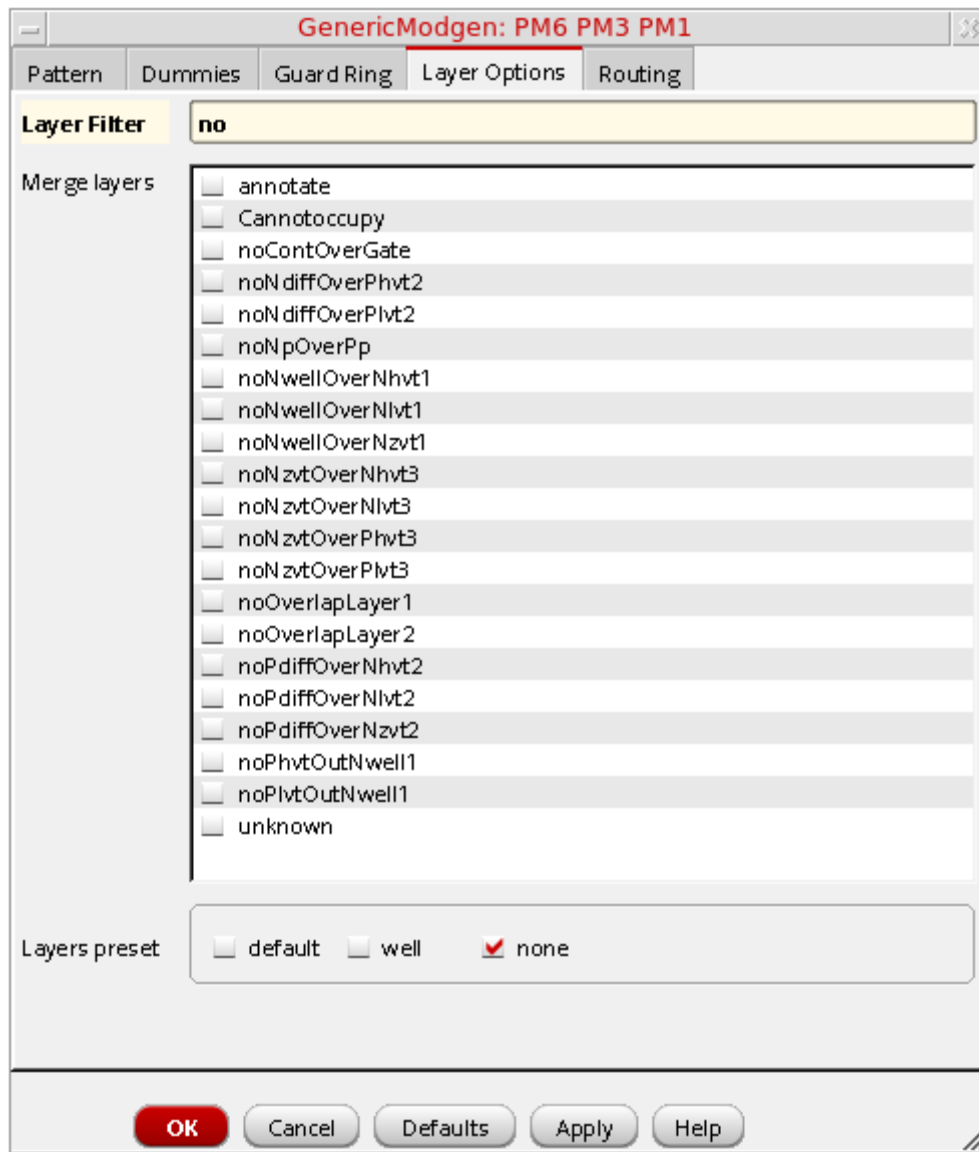
The *Layer Options* tab allows you to specify the *Merge layers* for the generated modgen by selecting the check box adjacent to the desired layers from the list or by selecting one of the *Layers preset*, that is, *default*, *well*, or *none*.



Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

You can filter the list of *Merge layers* by specifying the required criterion in the *Layer Filter* field. For example, if you specify `no` (as illustrated in the figure below) in the *Layer Filter* field, only the layer names that satisfy this filter criterion are listed in the *Merge layers* list box.



By default, the *Merge layers* list box in the *Layer Options* tab lists all available layers in the layout. However, if you select the *Use Layer Palette LPPs Only* check box in the Layout Editor Options form, it enables the synchronization of layers displayed on the *Layer Options* tab and the layers listed in the Palette assistant of Layout XL.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant



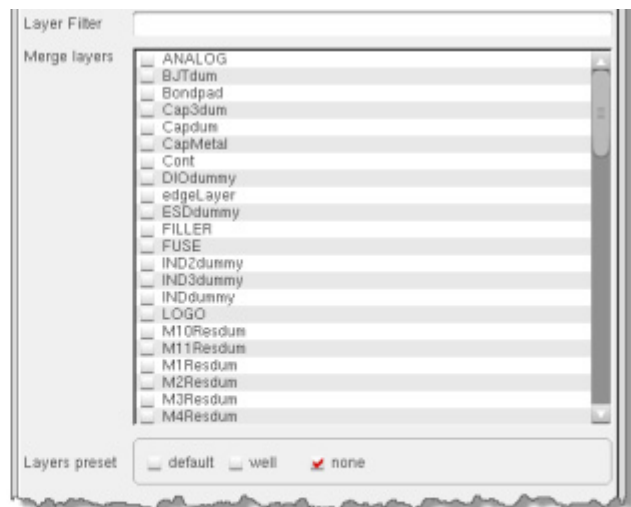
Tip

To access the Layout Editor Options form, choose the *Options – Editor...* menu from the Layout XL window.

When you enable layer synchronization and change the filters in the Palette assistant by using the available check boxes (that is, *Used*, *Valid*, and *Routing*), you can ensure layers of only specific category are displayed in the *Layer Options* tab's *Merge layers* list box. This change will be visible when you update the filters in the Palette assistant and then launch the GenericModgen form through the Constraint Manager, as illustrated in the figure below.



Before the Layer Palette LPP sync up
(note that the yellow entries are not visible in
the image on right)



After the Layer Palette LPP sync up

Specifying Standard Routing Options for Generated Modgen

The *Routing* tab can be used to specify the options to control the scope and region of pin to trunk routing for a generated modgen constraint.

The screenshot shows the 'GenericModgen: PM6 PM3 PM1' dialog box with the 'Routing' tab selected. The dialog has five tabs: 'Pattern', 'Dummies', 'Guard Ring', 'Layer Options', and 'Routing'. The 'Routing' tab contains the following options:

- Route**: ☐
- Trim Trunks**: ☒
- Horizontal Trunk Layer**: Metal2 (dropdown menu)
- Horizontal Trunk Width**: 0.08 (text field)
- Horizontal Trunk Spacing**: 0.07 (text field)
- Twig Layer**: Metal1 (dropdown menu)
- Twig Width**: 0.08 (text field)
- Twig Width Use Pin Width**: ☒
- Min Num Cuts**: 1 (text field)
- Add Vertical Trunks**: ☒
- Vertical Trunk Layer**: Metal3 (dropdown menu)
- Vertical Trunk Width**: 0.08 (text field)
- Vertical Trunk Spacing**: 0.07 (text field)
- Channel0 Device Spacing**: 0.14 (text field)
- Channel1 Device Spacing**: 0.14 (text field)
- Constraint Group**: virtuosoDefaultSetup (dropdown menu)

At the bottom of the dialog are five buttons: **OK** (highlighted in red), **Cancel**, **Defaults**, **Apply**, and **Help**.

The following steps explain the use of each option available on the *Routing* tab:

1. Select the *Route* check box to enable pin to trunk routing for the generated modgen.
2. The *Trim Trunks* check box is selected by default. This means that while routing, both the ends of a trunk will be trimmed to the first twig on that trunk. If you uncheck this check

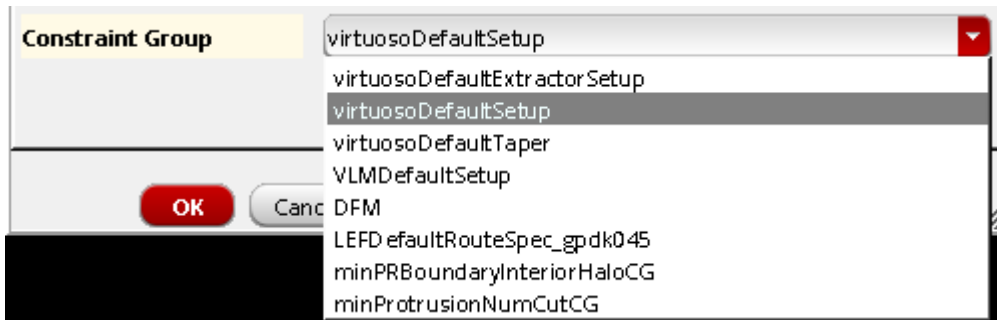
box, trunk trimming gets disabled and the length of the trunk will be the same as the width of the modgen.

3. Select a layer on which a horizontal trunk should be created from the *Horizontal Trunk Layer* drop-down list.
4. In the *Horizontal Trunk Width* field, enter a value that defines the width of the horizontal trunk when it is created.
5. In the *Horizontal Trunk Spacing* field, enter the required spacing between horizontal trunks in the same channel.
6. Select the *Twig Layer* to be used by the router for creating horizontal and vertical twigs.
7. Specify a value for the *Twig Width* for creating horizontal and vertical twigs.
8. The *Twig Width Use Pin Width* check box is selected by default. This means that the router will consider the width of the twigs equal to the width of the pin. If you deselect this check box, the router uses the width value specified in the *Twig Width* field.
9. The *Add Vertical Trunks* check box is selected by default. This means that a vertical trunk will be created for a net if a horizontal trunk exists for that net in multiple channels. If you deselect this check box, creation of vertical trunks gets disabled.
10. Select a layer on which a vertical trunk should be created from the *Vertical Trunk Layer* drop-down list.
11. In the *Vertical Trunk Width* field, enter a value that defines the width of the vertical trunk when it is created.
12. In the *Vertical Trunk Spacing* field, enter the required spacing between vertical trunks in the same channel.
13. Specify the *Channel0 Device Spacing*. This is the spacing between the last (bottom) trunk in the channel and the devices immediately below it.
14. Specify the *Channel1 Device Spacing*. This is the spacing between the first (top) trunk in the channel and the devices immediately above it.

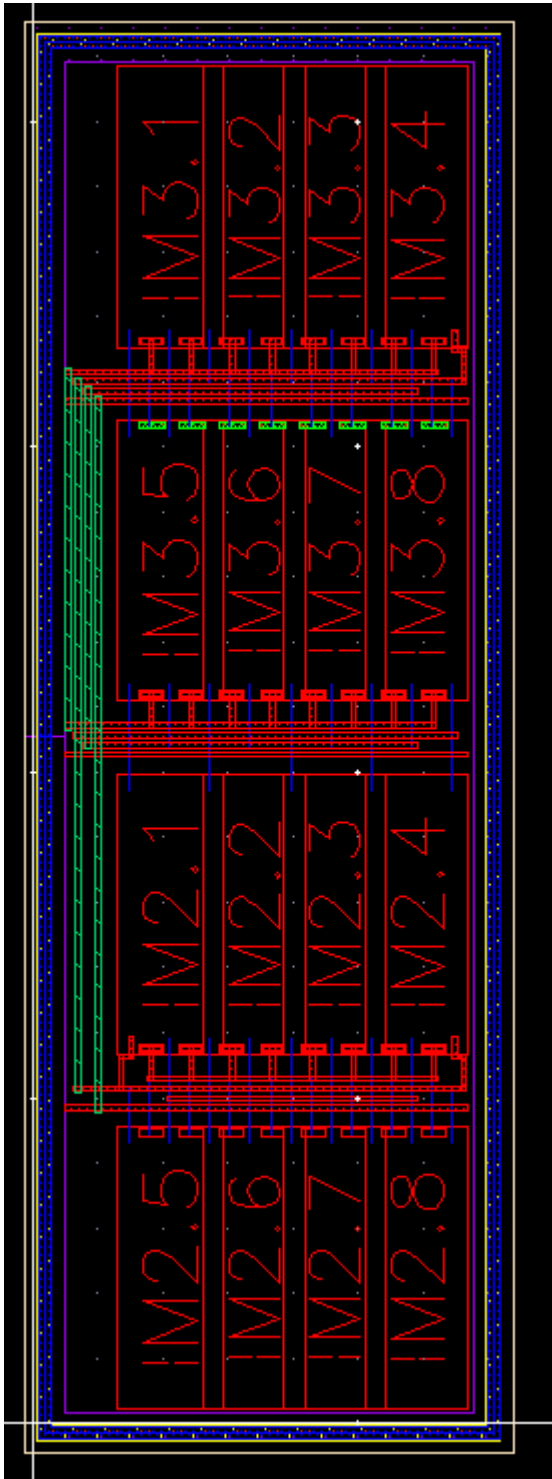
Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

15. Select the *Constraint Group* from the drop-down list to specify which routing rules should be used by default. By default, the *virtuosoDefaultSetup* constraint group is selected.



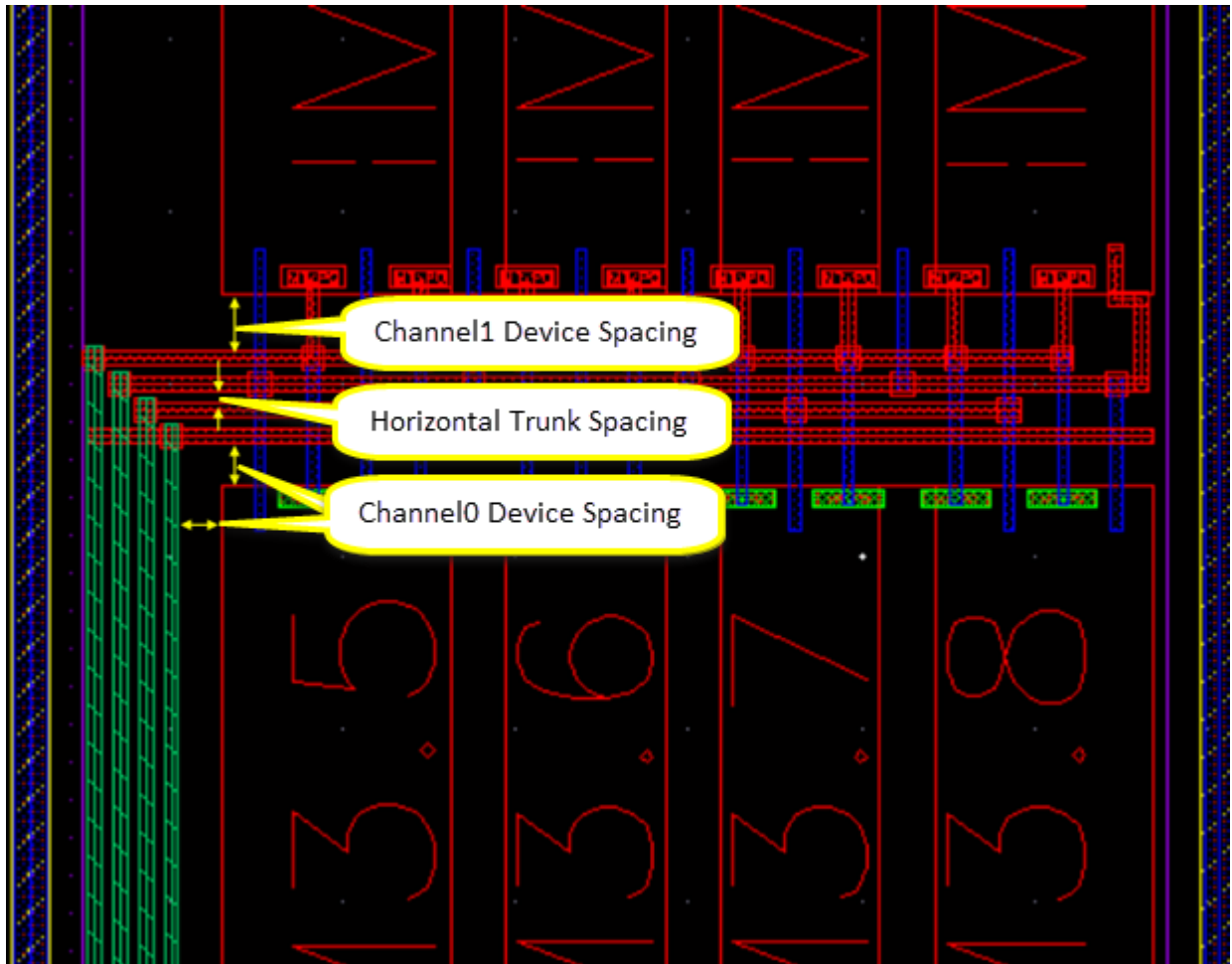
The figure below illustrates a routed modgen:



Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

The figure below shows the different types of device spacings in the routed modgen illustrated above:



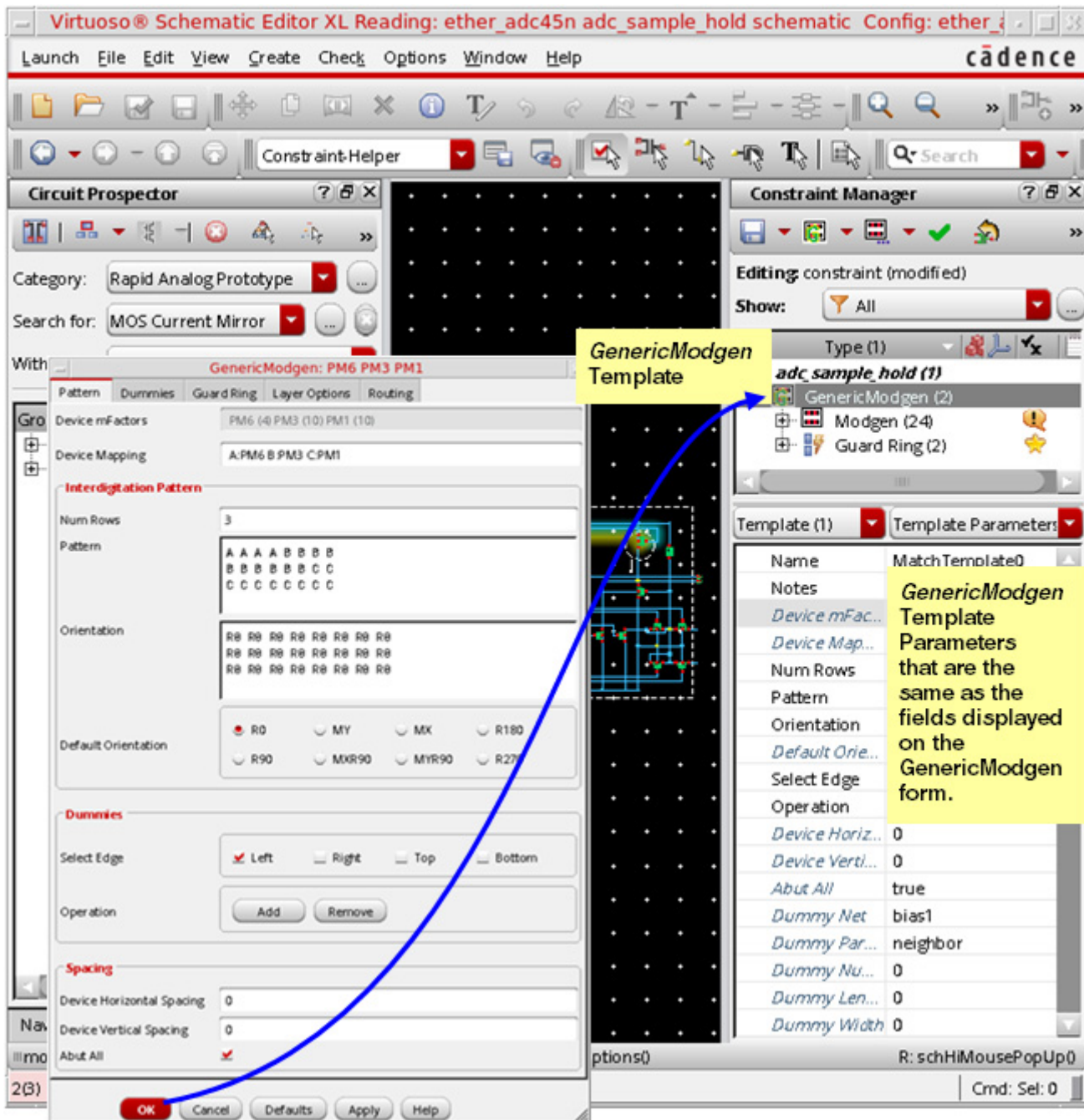
Generating a GenericModgen Template

After specifying the required configuration information on all the tabs of the GenericModgen form, click **OK**. This generates a *GenericModgen* template in the Constraint Manager

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

assistant that contains a modgen with the specified configuration, as shown in the figure below.



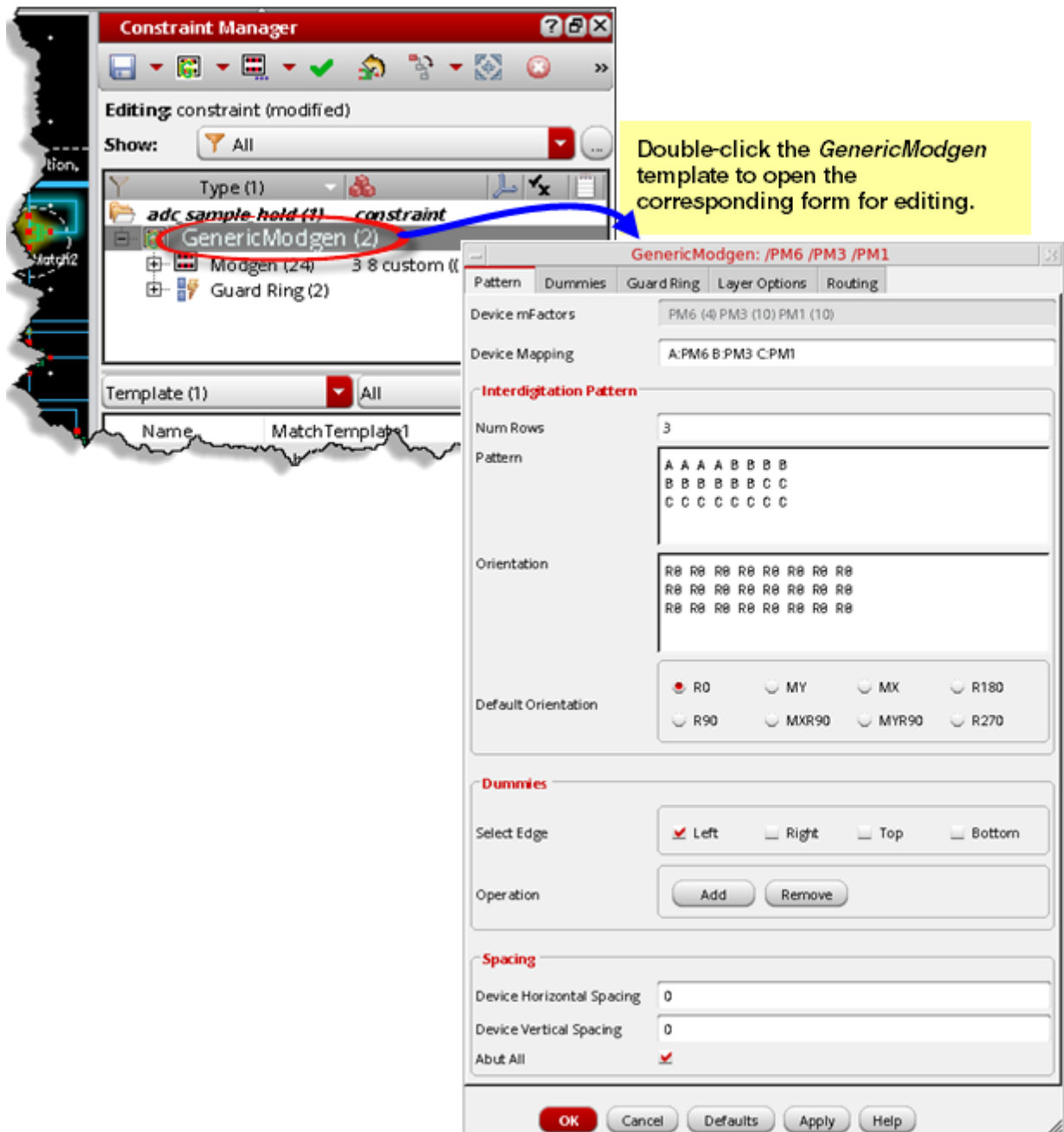
Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Editing a Generated Modgen (GenericModgen Template)

To edit the generated *GenericModgen* template, do one of the following:

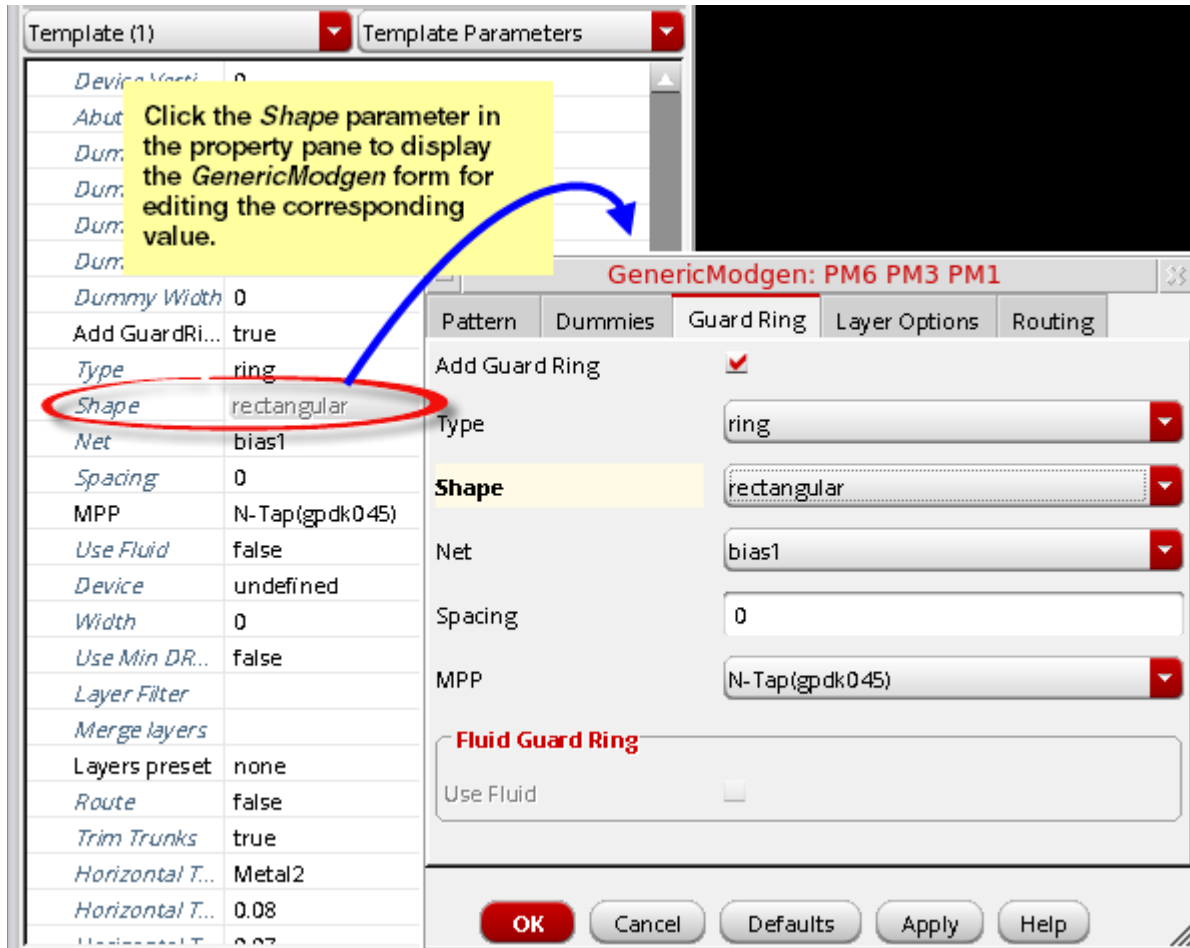
- Double-click the *GenericModgen* template in the *Constraint Browser*. This action launches the *GenericModgen* form with no fields selected and the *Pattern* tab displayed, as shown below.



Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

- Click the parameter value you want to edit in the Constraint Manager assistant's property editor pane. This action opens the *GenericModgen* generator form with that specific parameter field highlighted and in focus. If the parameter is not on the *Pattern* tab, then the appropriate tab is displayed to make the parameter visible.



Viewing or Editing a Generated Modgen (GenericModgen Template)

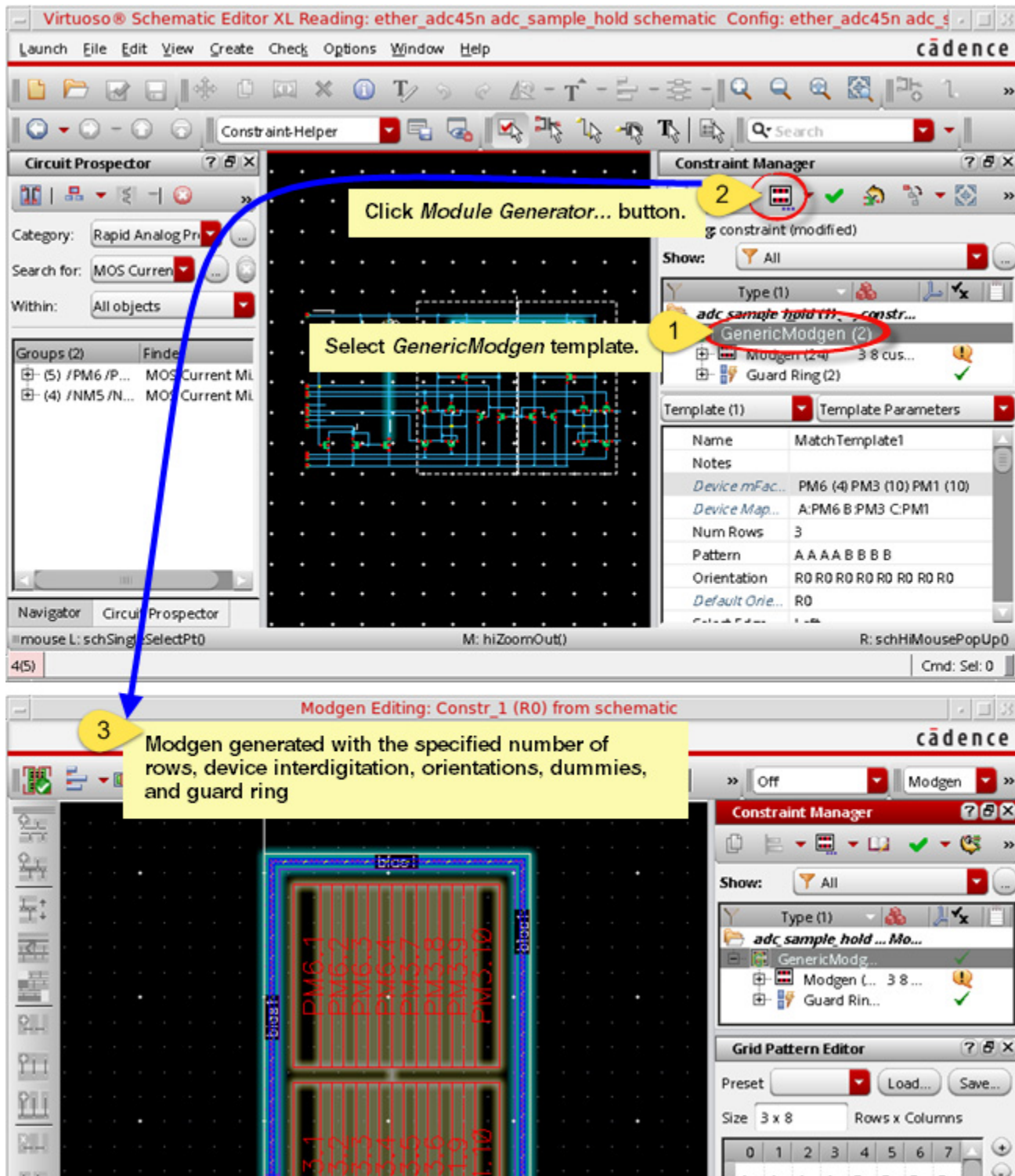
A generated modgen can be previewed and edited within the Modgen Editor. To do so,

1. Select the *Modgen* or *GenericModgen* template in the Constraint Manager.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

2. Click the *Module Generator...* button on the Constraint Manager toolbar, as shown below.



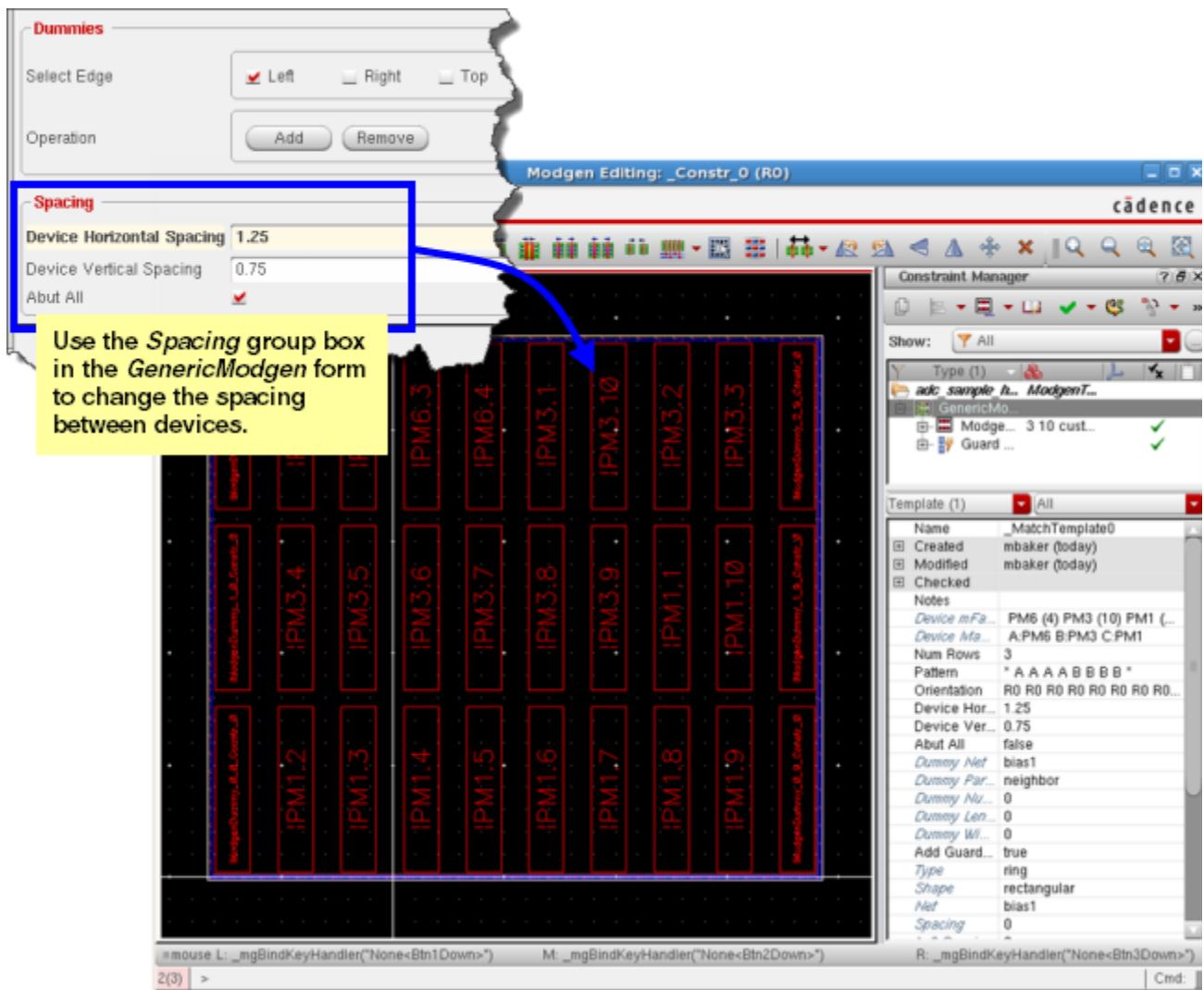
Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

In the Modgen Editor, you can modify the values specified for *Device Horizontal Spacing* and *Device Vertical Spacing* of the generated modgen, and deselect the *Abut All* check box. Refer to the figure below to see an illustration of such edits.

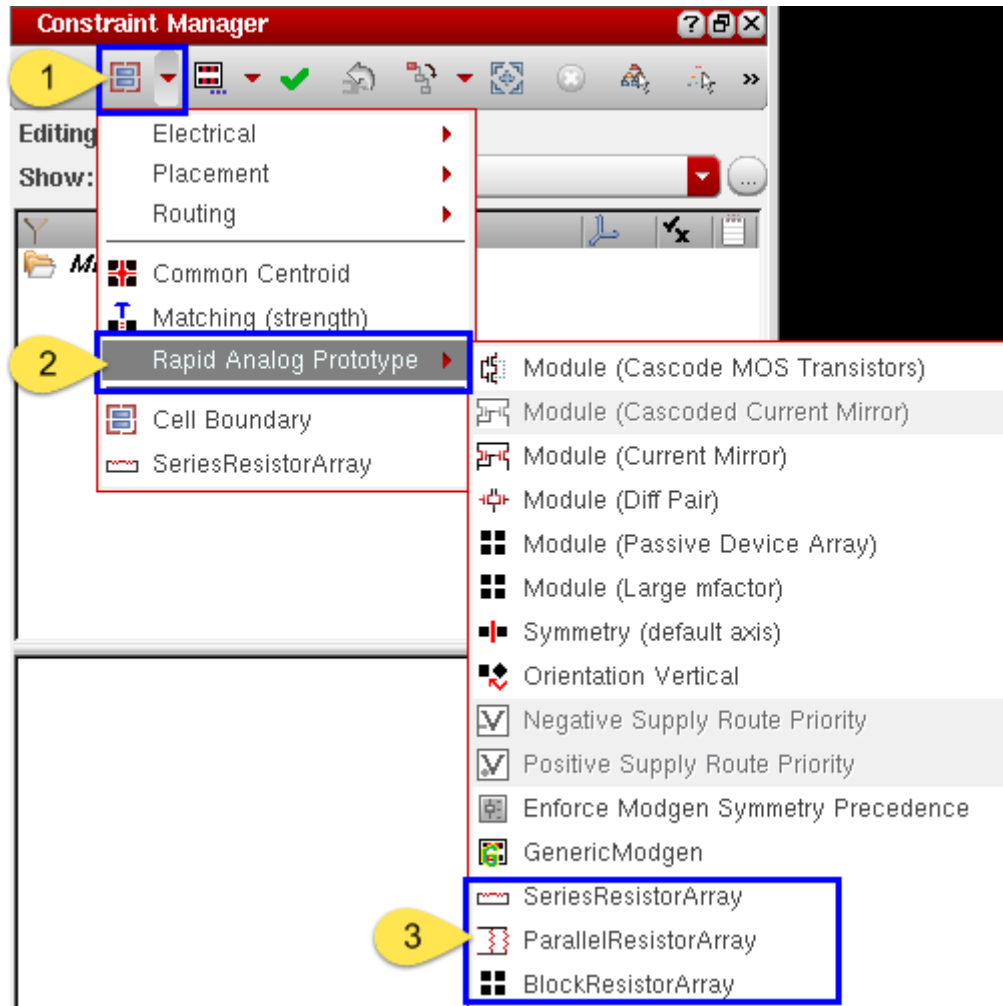


The generated modgen should only be modified through the GenericModgen template parameters. If it is edited using the Modgen Editor, the GenericModgen template parameters will be out-of-sync with those edits. Also, those edits will be lost if the GenericModgen template parameters are subsequently changed.



Creating Modgen Template for Resistor Arrays

The *Rapid Analog Prototype* submenu on the *Constraint Creation* menu of the Constraint Manager assistant provides the following three constraint generator options: *SeriesResistorArray*, *ParallelResistorArray*, and *BlockResistorArray*. You can use these options to create a modgen template for arrays of resistors.



Note:

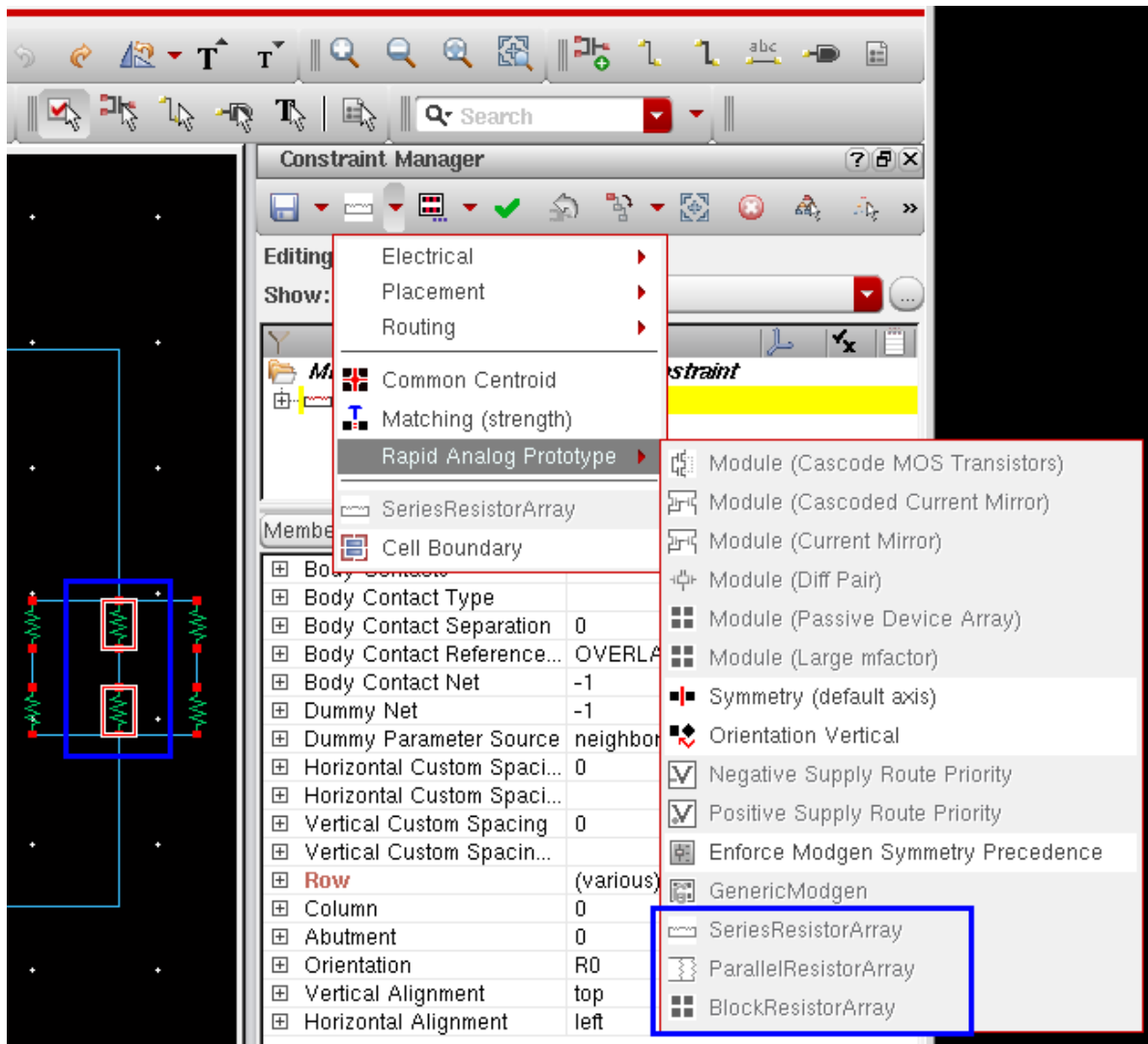
- ❑ To create a resistor array template, it is necessary that all the selected devices are resistors; otherwise, an error message is displayed in the CIW and the resistor array template is not created.
- ❑ Block resistor arrays can also be generated on resistors that are shorted and retained in the design for future consideration. These shorted resistors are arranged

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

as per their order of position so that in the future, if they are unshorted due to design change, then the whole configuration need not be re-generated.

- ❑ The *SeriesResistorArray*, *ParallelResistorArray*, and *BlockResistorArray* constraint generator options appear as disabled (grayed out) till you select at least one device in the schematic or layout canvas.
- ❑ If the selected set of devices contains even one device on which a modgen constraint was created previously, the *SeriesResistorArray*, *ParallelResistorArray*, and *BlockResistorArray* constraint generator options appear as disabled (grayed out), as shown below.



To create a modgen template for a particular type of resistor array, click the corresponding option on the submenu. The form for that resistor array type appears. The fields in all three

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

forms are the same. The only difference is in the name of the form, which corresponds to the chosen resistor array constraint generator option. For example, if you chose the *SeriesResistorArray* constraint generator option to create a resistor array constraint on resistors R7 and R10, the following form is displayed.

SeriesResistorArray: /R7 /R10

Row: 2

Col: 1

Horizontal Spacing: 0

Horizontal Spacing Layer: Drc

Vertical Spacing: 0

Vertical Spacing Layer: Drc

Topology: [Grid of 12 icons]

Place Rotation: V

Add Routes: ☒ No ☐ Yes

Route Width: 0.06

Route Layer: Metal1

Dummy Net: net22

OK Cancel Defaults Apply Help

In the displayed resistor array constraint generator form, set the parameters of the required resistor array constraint as follows:

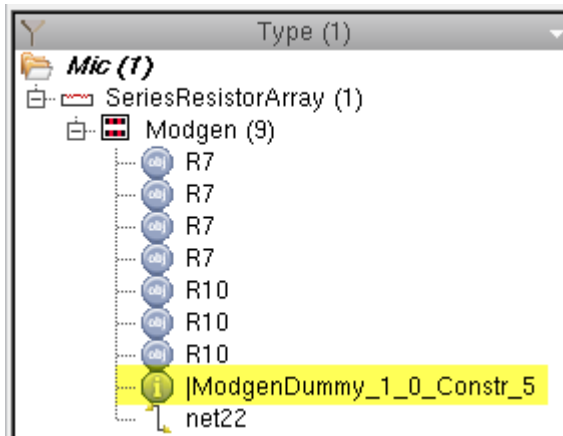
1. Specify the number of *Row* or *Col* (column).

By default, these two fields display values that are calculated automatically based on the segments of the selected resistors taking into consideration the most optimal, rectangular, fully-filled pattern for the generated resistor array. The same consideration drives the recalculation of either the *Row* or *Col* field, if one of these two fields is modified. If the recalculated value is not an integer, the value gets rounded off to the closest higher number, which can cause dummies and spaces to be added to the

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

generated resistor array. For example, if the selected resistors R7 and R10 have 4 and 3 segments, respectively, the values displayed by default in the *Row* and *Col* fields in the resistor array constraint generator form are 1 and 7, respectively. If you now change the value in the *Row* field to 2, the *Col* field is automatically updated to 4. When you generate the modgen template for the selected series resistor array, a dummy resistor array constraint is also added, as shown below.

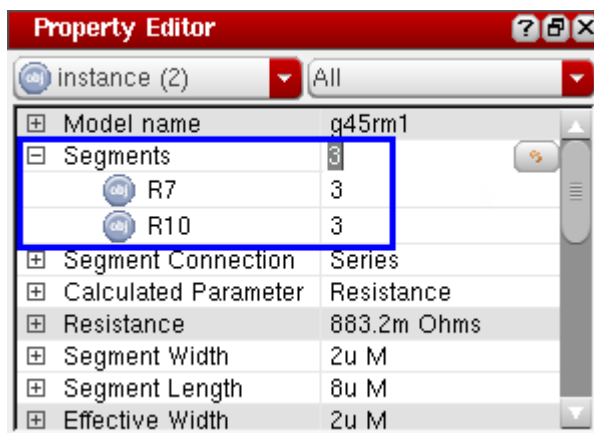


The resistor array generators avoid any dummy usage unless the array fails to balance the configuration. Based on the selected topology and rotation, a resistor array can achieve single row/column configuration especially for series/parallel configuration in the *BlockResistorArray* constraint generator.



Tip

While generating a resistor array template, you can change the value in the *Segments* field for the selected devices in the Property Editor assistant.



If you are unable to edit this field, do one of the following:

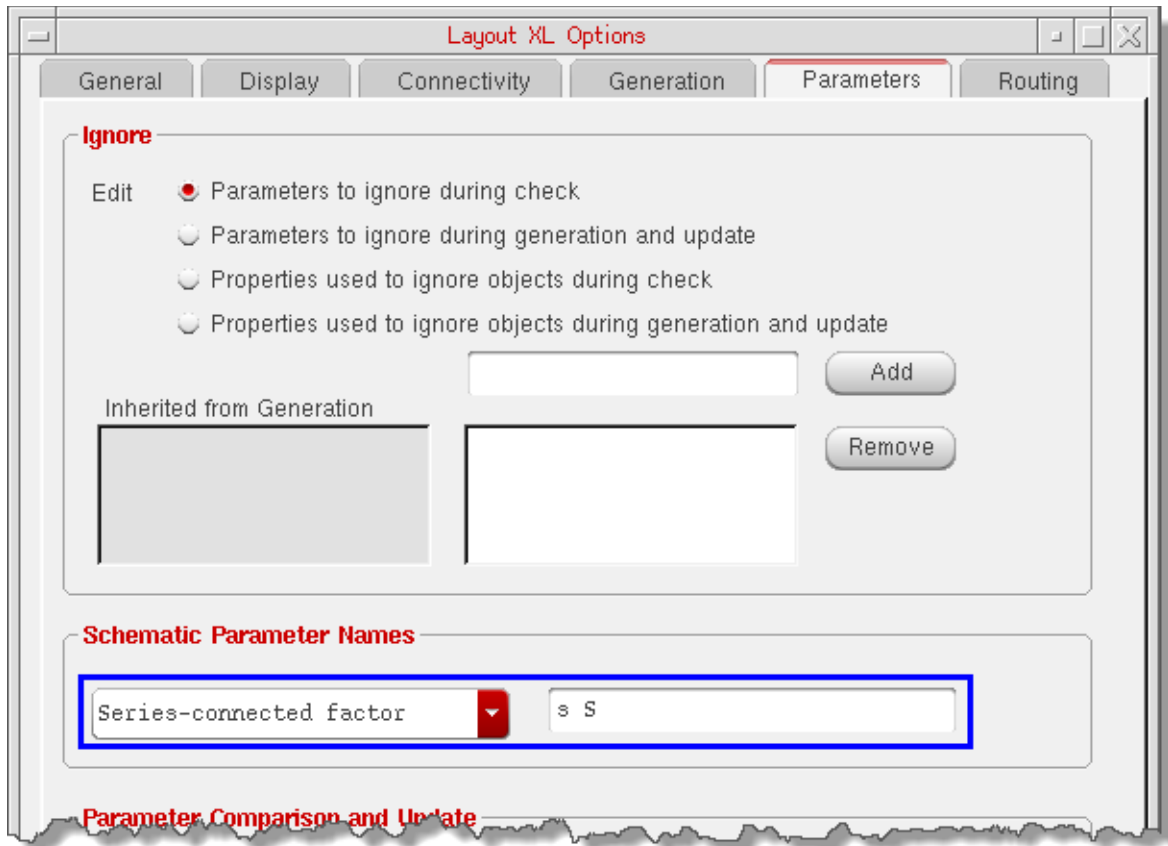
Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

- ❑ Set the `sfactorNames` Layout XL environment variable. For example, in the CIW, run the following command:

```
envSetVal "layoutXL" "sfactorNames" 'string "s S segments"
```

- ❑ In the Layout XL Options form (*Options - Layout XL*), choose *Series-connected factor* from the drop-down list box in the *Schematic Parameter Names* group box, as shown below.

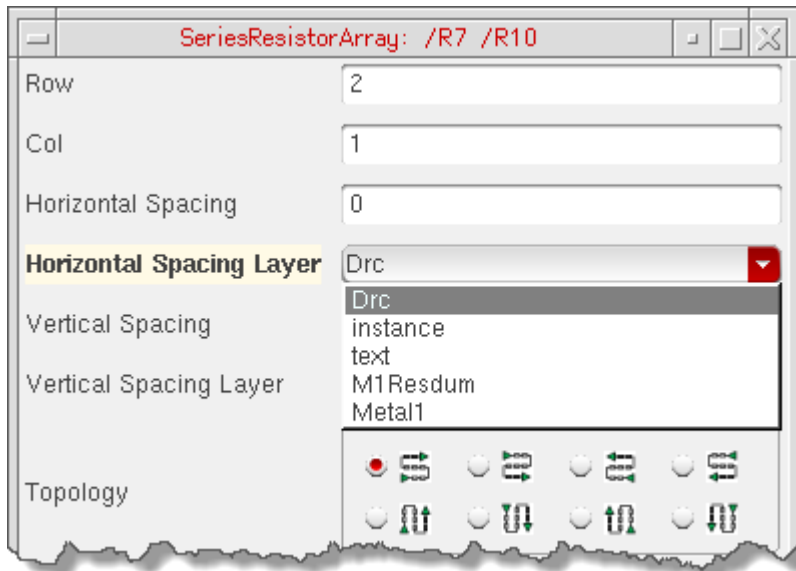


2. To continue with setting the parameters on the resistor array constraint generator form, specify the *Horizontal Spacing*. This value is used to identify the spacing required between columns of devices in the generated resistor array.
3. Choose the *Horizontal Spacing Layer* from the available options, as shown below. This list is populated with the names of the layers available in the layer headers of the corresponding layout cellview. By default, the spacing between devices is according to

Virtuoso Unified Custom Constraints User Guide

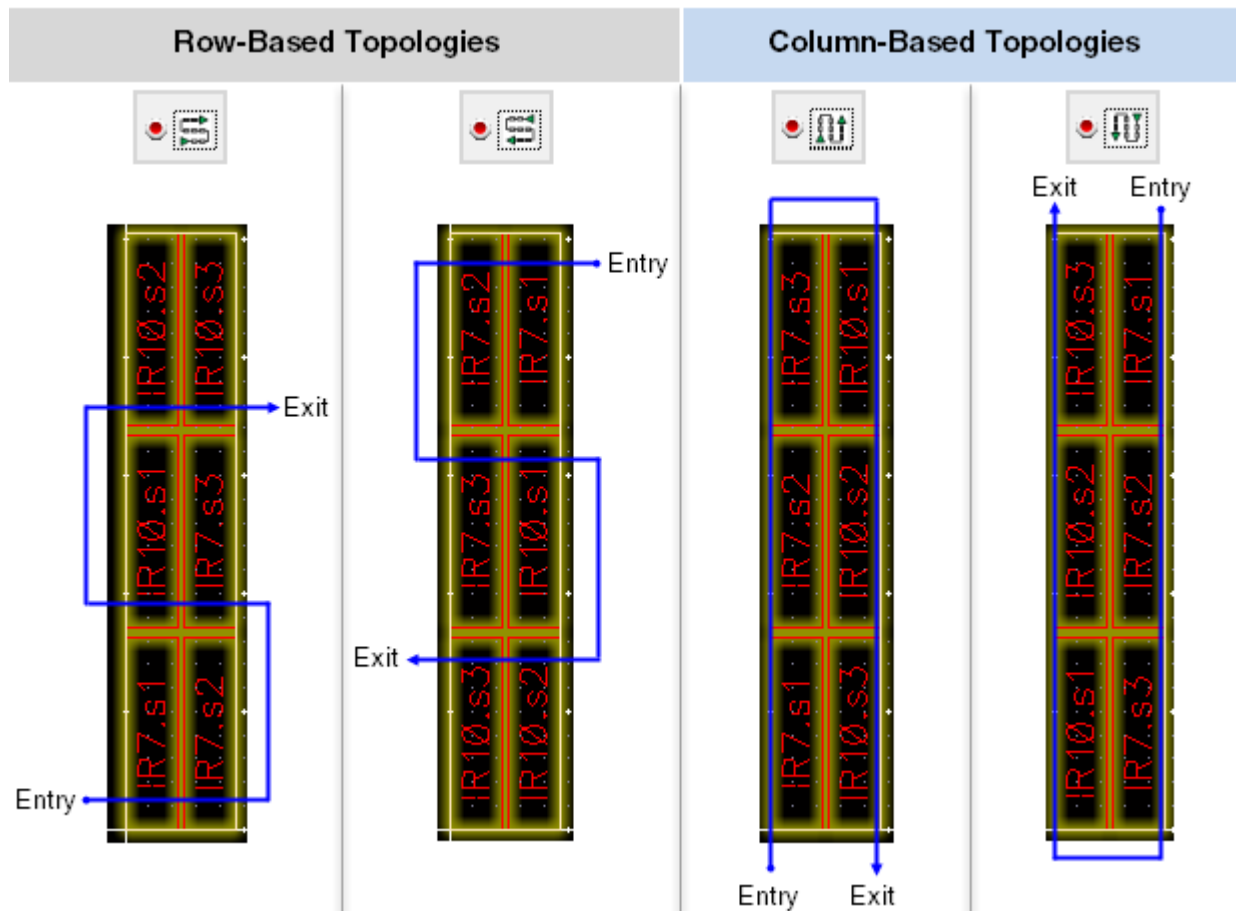
The Constraint Manager Assistant

all the design rules (*Drc*). By selecting *Drc* from the *Horizontal Spacing Layer* list, you can avoid the requirement of setting any spacing constraint.



4. Specify the *Vertical Spacing*. This value is used to identify the spacing required between rows of devices in the generated resistor array.
5. Choose the *Vertical Spacing Layer* from the available options. This list is the same as the *Horizontal Spacing Layer* drop-down list box.
6. Select the *Topology* for the generated resistor array from the available eight serpentine topologies. Out of these eight, four are row-based and four are column-based. The selection of a topology can help you change the orientation of the resistors in the

generated resistor array. The following figures illustrate how topology is translated in a resistor array:



7. Specify the *Place Rotation* for the generated resistor array as *H* (horizontal) and *V* (vertical). If the `figs` associated with the pins of the resistor are left and right, the default *Place Rotation* is *H*; if the `figs` are top and bottom, the default *Place Rotation* is *V*.
8. Set *Add Routes* to *Yes* if routing needs to be added to the generated resistor array. By default, the *No* radio button is selected, which implies that adding of routes is disabled.
9. Specify the *Route Width* and *Route Layer* if you choose to add routing to the generated resistor array.
10. Select the *Dummy Net* that should be used in the modgen if the modgen contains dummy devices. This list contains the names of all the nets associated with the selected devices.
11. Click OK to generate the modgen template for the selected resistors.

Creating Constraints at Different Design Levels

In addition to creating constraints at the top level of design (see also [Creating Constraints in the Constraint Manager](#)), you can also create constraints further down the design hierarchy.



Tip

By default, constraint directories are *off*. To activate these directories, right click the *Constraint Browser*, select *Options*, then ensure that *Show Folders* is selected in the displayed Constraint Manager Options form.

If you descend down a level in the design hierarchy, (using for example, *Edit – Hierarchy – Descend Edit*) and open the descended cell in a *New Tab*, you will notice that the *Constraint Manager* will clear itself of any constraints that had been created at any level in the hierarchy above the current schematic view. The *Constraint Manager* will only of course be empty if no constraints have been previously created at the level of the design now on display.

Note: When a configuration is controlling the current design hierarchy (that is, the current schematic has been opened in the context of a CPH (see [Configure the Physical Hierarchy](#)) or a HED configuration), and you then descend into a schematic that is on a path outside of that hierarchy, the *Constraint Manager* will consequently not be able to compute a constraint view because the path is not now part of that managed hierarchy. In such cases, the *Constraint Manager* will display a message under the toolbar area informing you that you are: “Currently below the design hierarchy controlled by the physical implementation. Open the *Configure Physical Hierarchy* tool from the layout window to modify the design hierarchy”.

Note: You are unable to generate constraints from within a Pcell cellview.

The *Constraint Manager* only displays those constraints that have been created in the current cell’s constraint view and below. Constraints are always created in the constraint view for the current cell.

If you descend a further level down into the design hierarchy and create some constraints, these constraint will still be visible in the *Constraint Manager* if you again return back up a level.

- Any constraints created at lower levels of the hierarchy are stored in constraint views, as described in [Constraint Storage](#).
- Any constraints that are stored in the constraint view that you are editing for the current constraint view (see [The Constraint View](#)) will appear with a white background.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

- Any constraints that exist in levels further down the design hierarchy (that is, constraints that are not stored in the current constraint view) will appear with a gray background.
- The *Constraint Manager* displays constraints for all schematics in the hierarchy, from the schematic currently being edited down to the stop points of the current configuration.

Note: In layout, the *Constraint Manager* re-routes as you descend the layout hierarchy, in effect considering the newly descended cell as the root cell.

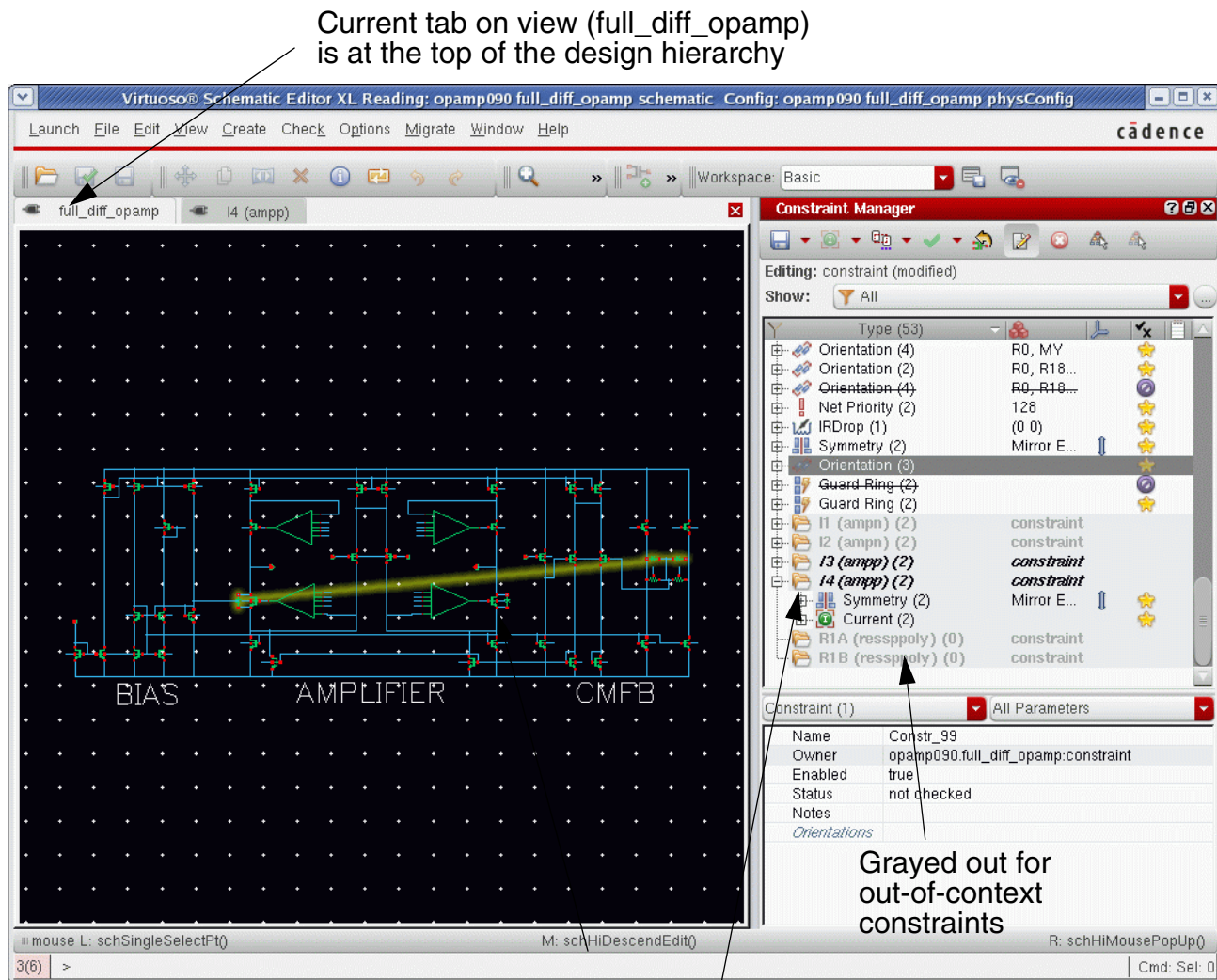


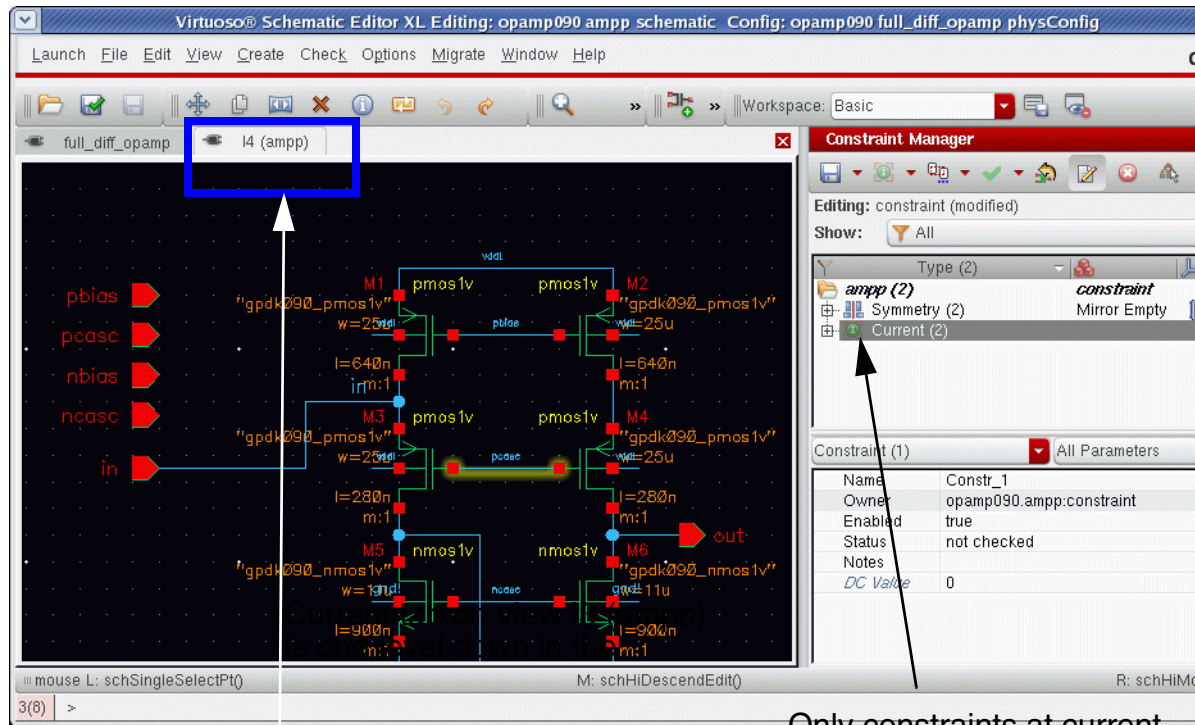
Figure 1-13 Constraints shown created at different levels of the design hierarchy

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

If we now switch tabs, to the I4 (amp) view, you will notice that the full_diff_opamp constraint view constraints are now not shown as they were created at the level **above** I4 (amp).

Note: As mentioned, the *Constraint Manager* only shows those constraints created at the current level and below.



I4 (amp) tab selected

Only constraints at current level or below are shown. No constraints created at the higher full_diff_opamp level are shown.

Figure 1-14 Constraint created at the current level

When the constraints are transferred to the layout view (see [Transferring Constraints Between Schematics and Layout](#)), *ALL* constraint views for the entire design are transferred to the layout view, not just the top level constraints.

Note: Full hierarchical cross-selection is also possible between the *Constraint Manager* in layout and its equivalent in the schematic at all levels in the design hierarchy.

Viewing Constraints in the Constraint Manager

You can view all constraints that have been created in the cache, irrespective of whether they were solely created for back-end or front-end use.

For example, a Correlation constraint that has been created in the front-end, for front-end use only, will still be visible in the *Constraint Manager* in the back-end. Constraints are therefore never hidden, irrespective of whether they can be used in their current context or not.

See also:

- [Transferring Constraints Between Schematics and Layout](#)
- [Updating Constraints](#)

Moving Constraints to Current or Sub-Cells

A constraint will be shown multiple times in the *Constraint Manager* if a particular constraint object exists at numerous lower levels in the design hierarchy.

A constraint directory for each instantiation of that object will be displayed in the *Constraint Manager*. That is, a constraint directory will be displayed for each instance that the object is found in the design below the current point.

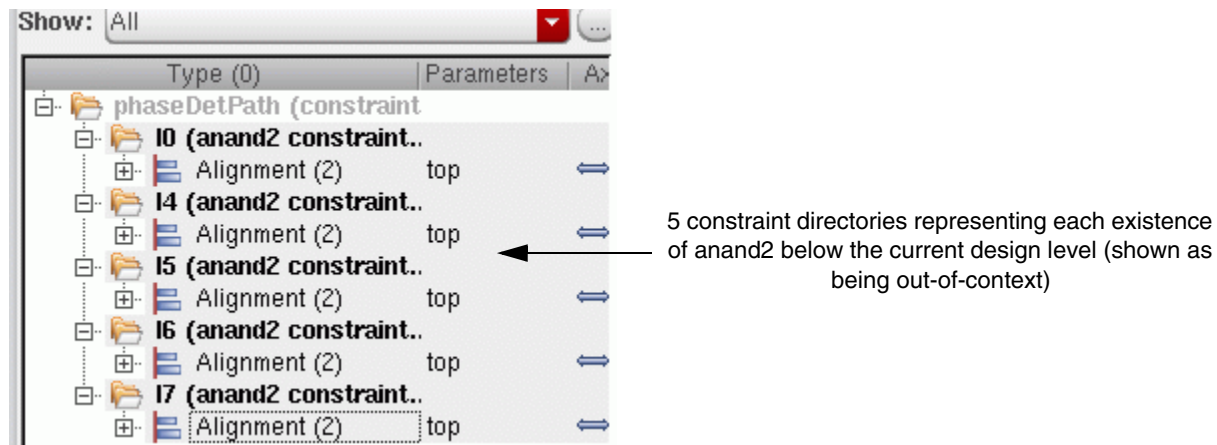


Figure 1-15 Multiple existences of anand2 exist below this level in the design

You can however choose to change this so that the constraint only exists at the current level in the design hierarchy.

Moving a Constraint to the Current Cell



To be able to move an existing constraint to the current cell on view, you need to have a hierarchical cell with constraints that have already been set in the sub-cell in question.

If you have created a constraint in a sub-cell (that is, a cell this exists lower in a hierarchy), you can move that constraint to exist only in the top cell.

To move a constraint to the current cellview, perform the following steps:

Note: Steps 1-3 are optional depending on whether the *Move to Current Cell* option has already been added to the *Constraint Browser* context menu.

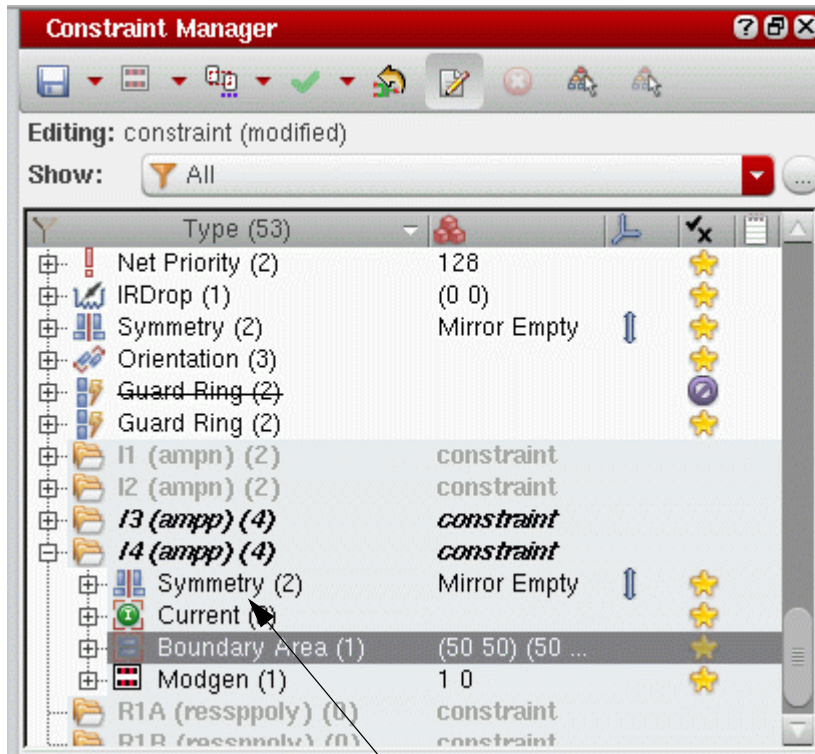
1. Right-click in the *Constraint Browser* area to display the context-menu.
2. Select *Options* to display the Constraint Manager Options form.
3. Ensure that in the *Advanced Options* section the *Enable Move Constraints to/from Current Cell* check box is selected, then click *OK* in the Constraint Manager Options form.
4. Ensure that the cellview that you want to move the constraint to is current.
5. In the *Constraint Browser*, select the (currently out-of-context) constraint that is to be moved to the current cell.

Note: The constraint to be moved to the current cell must be a constraint that was created in a (lower) sub-cell in the hierarchy. You must also currently be at the (top) level

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

of the hierarchy at which you want the sub-cell to move to.



(Out-of-context) constraints in cell I4(ampp)
Note location of the Symmetry constraint

6. Right-click over the *Constraint Manager* and select *Move to Current Cell* from the displayed Constraint Browser Context-Menus.

Note: The status bar displays information on what the *Move to Current Cell* option

does and, where appropriate, the reason why it may not be enabled at a particular time.

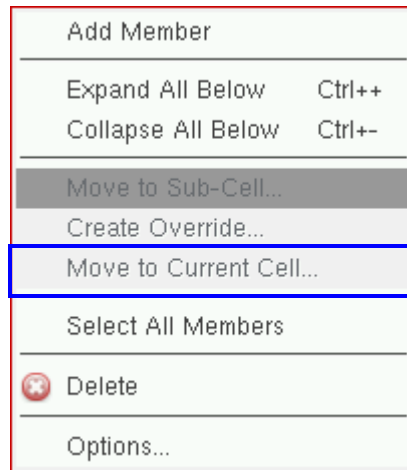


Figure 1-16 Move to Current Cell in the Constraint Manager Context-Menu

A confirmation message will be displayed asking you to confirm your action. You will be advised that all constraint views will have to be saved following the operation. If only one constraint has been selected, the constraint view to be saved will named.

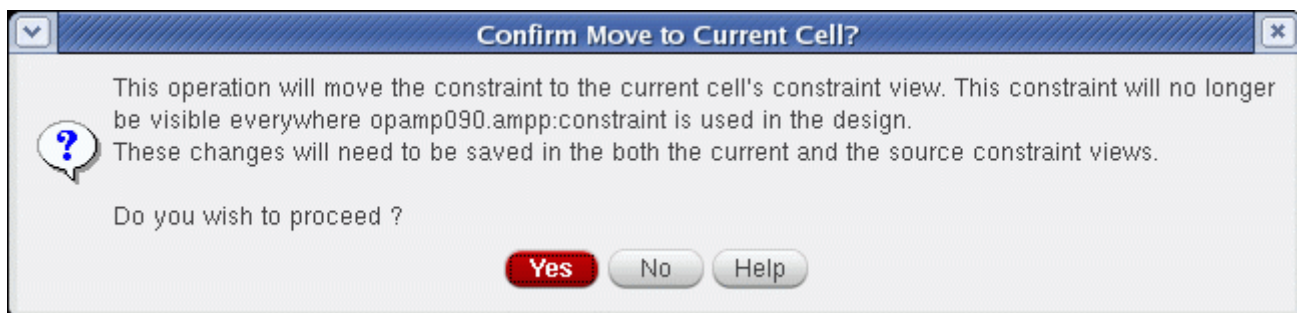


Figure 1-17 Confirm Move of Selected Constraint to the Current Cell

7. Select *Yes* to confirm that the selected constraint be moved to the current cell only.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

The constraint now only exists in the current cell (see also the *Owner* constraint attribute value in the *Constraint Parameter Editor* pane).

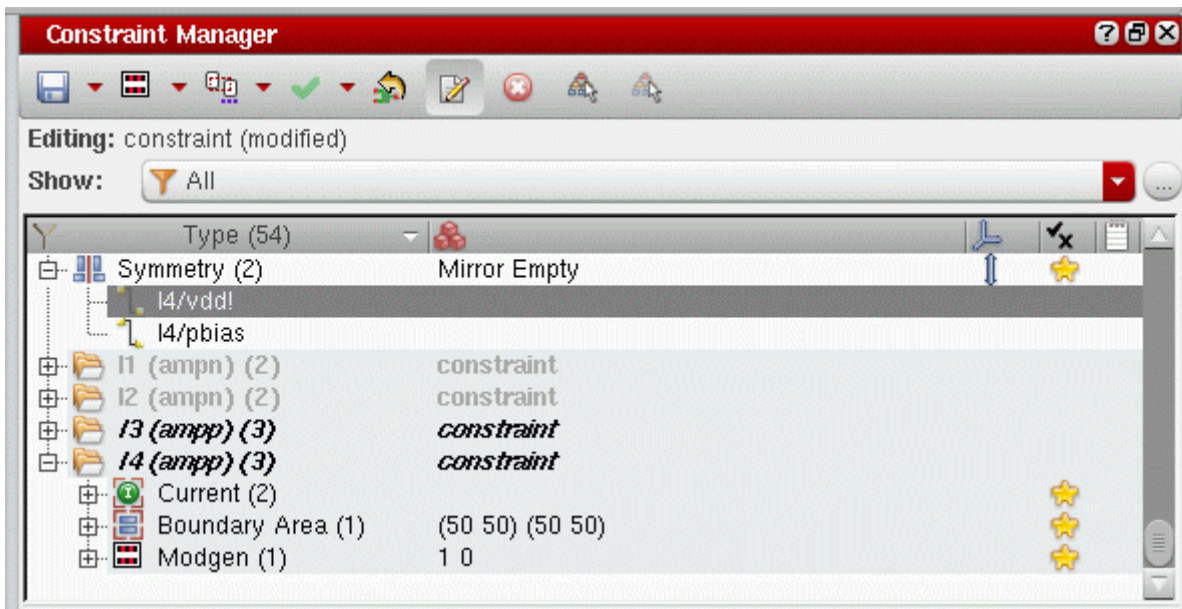


Figure 1-18 Symmetry Constraint Moved to Current Cell

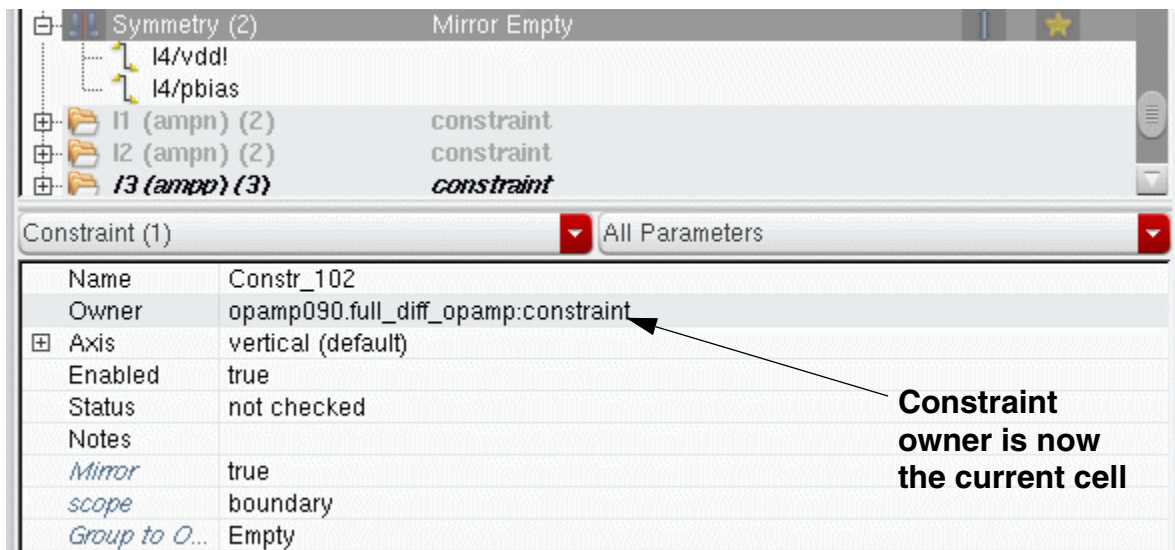


Figure 1-19 Constraint Owner attribute updates for new location of constraint

Note: For each selected constraint, a message will be displayed in the CIW detailing the source and destination constraint views.

See also [Moving a Constraint to Sub-Cells](#) and [Overriding Constraints](#).

Moving a Constraint to Sub-Cells

You can choose to move a constraint that has been created in the top cell to the sub-cellviews of its existing members.

For example, a constraint created in top cell `full_diff_opamp` with member instances `I3` and `I4` will initially, after creation, exist only in the cellview it was created in. That is, `full_diff_opamp`. However, if you then choose to move that constraint to the member sub-cells, it will no longer exist in `full_diff_opamp`, rather it will now exist in multiple locations in instance cellviews `I3` and `I4`. Consequently, it will now be viewable in the *Constraint Browsers* of these cellviews (`I3` and `I4`) and viewable as out-of-context (that is, the constraint is in the same hierarchy, but as part of a different traversal) in the top cellview (`full_diff_opamp`).

To move a constraint to sub-cells below the current top cell view:

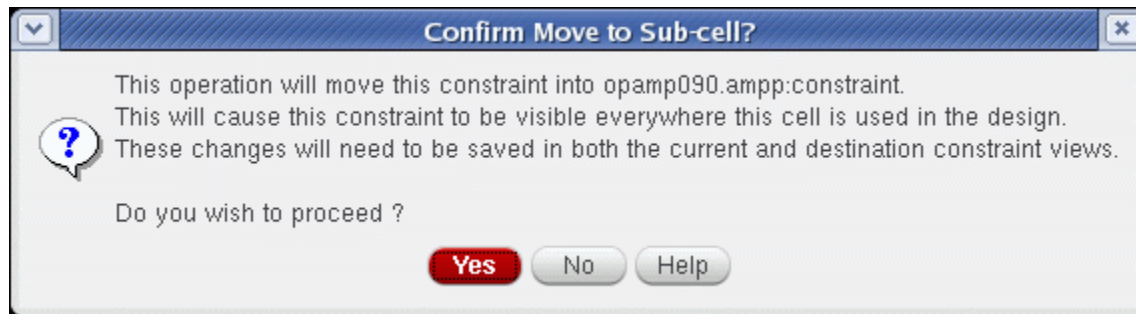
Note: Steps 1-3 are optional depending on whether the *Move to Sub-Cell* option has already been added to the *Constraint Browser* context menu.

1. Right-click in the *Constraint Browser* area to display the context-menu.
2. Select *Options* to display the Constraint Manager Options form.
3. Ensure that in *Advanced Options* the *Enable Move Constraints to/from Current Cell* check box is selected, then click *OK* in the Constraint Manager Options form.
4. Ensure that the cellview containing the constraint that you want to move to sub-cells is current.
5. Click the constraint in the *Constraint Manager* that you want to move to its member sub-cells.
6. Right-click over the *Constraint Manager* and select *Move to Sub-Cell* from the displayed Constraint Browser Context-Menus.

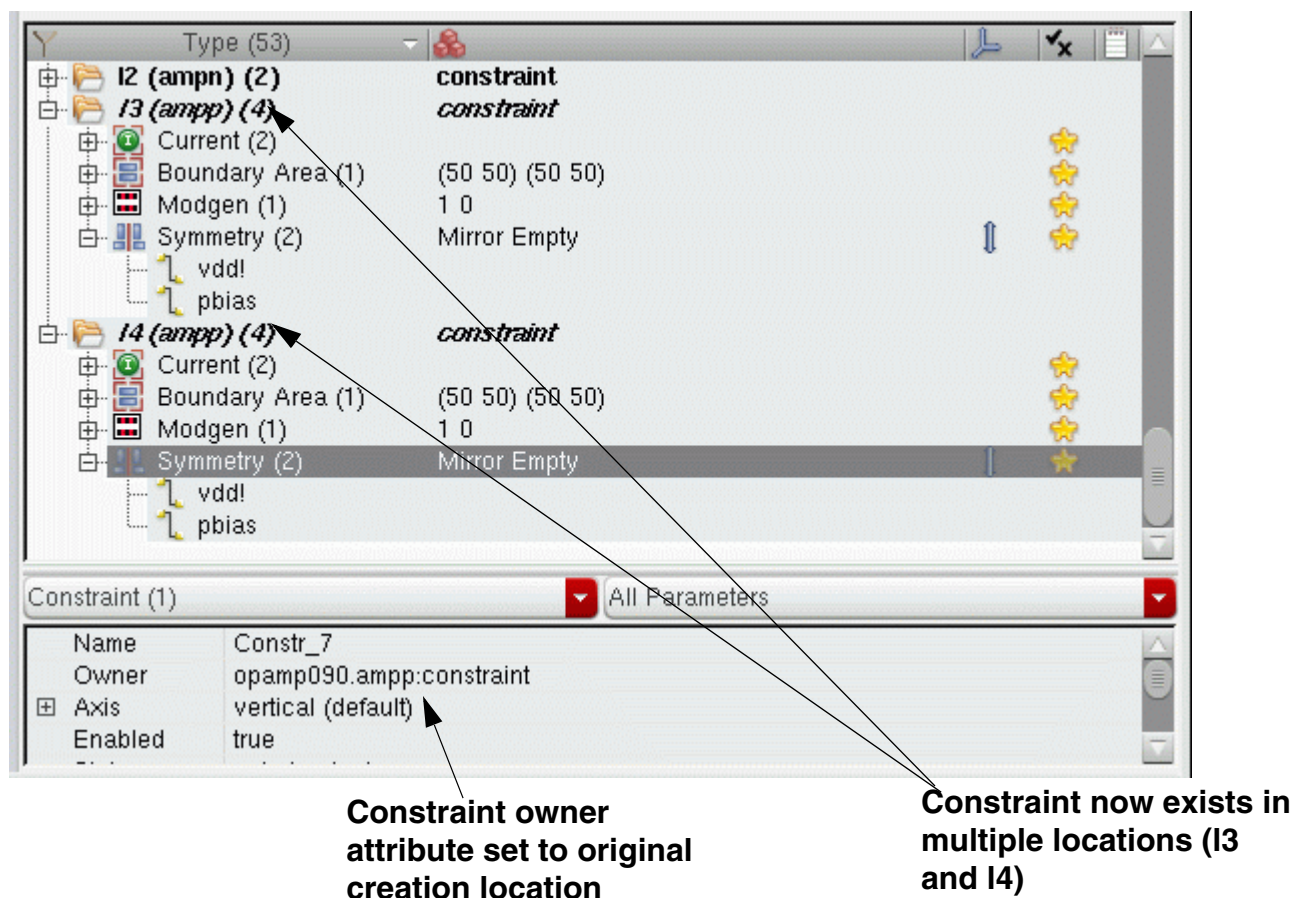
Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

A confirmation message will be displayed asking you to confirm your action. You will be advised that all constraint views will have to be saved following the operation. If only one constraint has been selected, the constraint view to be saved will named.



7. Select **Yes** to propagate the constraint everywhere that the current cell is used in the design. The constraint will now become a multiple occurrence constraint, existing at sub-cell levels.



Note: For each selected constraint, a message will be displayed in the CIW detailing the

source and destination constraint views.

See also [Moving a Constraint to the Current Cell](#) and [Overriding Constraints](#).

Overriding Constraints

Constraint overrides provide the option of having a constraint set as *overridden* rather than deleted or modified. Setting a constraint as overridden can allow, for example:

- Layout constraints to differ from schematic constraints.
For example, the ability to define a constraint on the physical side that will override or disable a constraint on the logical side.
- Variants of constraints to exist.
For example, the ability to define a constraint at the top of the logical design hierarchy that overrides or disables a constraint that has been inherited from lower levels of a hierarchy.

You can therefore use a constraint override to create an override of a lower level constraint in the current cell's constraint view. Creating the override will create a copy of the constraint at the current level.

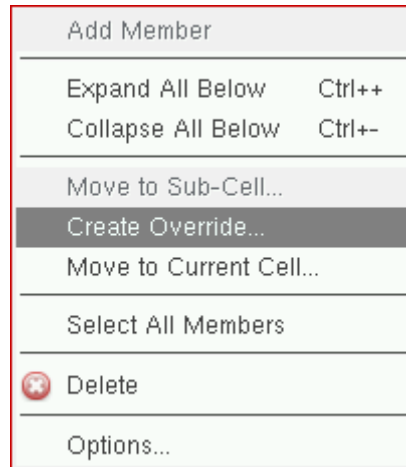
The *Create Override* option can be added to the *Constraint Browser* context-menu by:

1. Right-click over the *Constraint Browser* to display the *Constraint Manager* context-menu.
2. Select *Options* to display the Constraint Manager Options form.
3. Check the *Enable Constraint Overrides* check box (this may need to be preceded by first selecting the *Show Folders* check box).

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

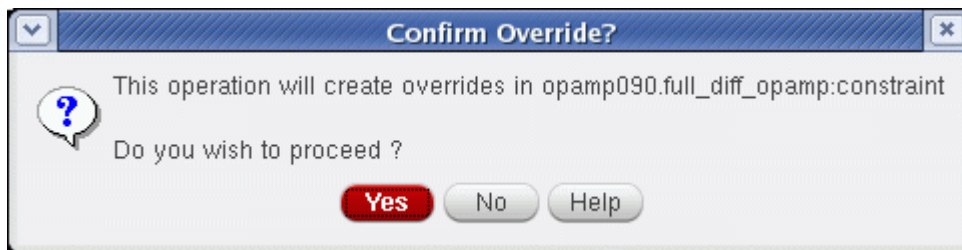
4. Click *OK* to close the Constraint Manager Options form and have the *Create Override* option added to the *Constraint Browser* context-menu.



To create a constraint override:

1. Select the constraint that you want to override in the *Constraint Manager*.
2. Select *Create Override* from the Constraint Browser Context-Menus (if this option is not available, see above on how to add it).

A confirmation message will be displayed asking you to confirm your action.

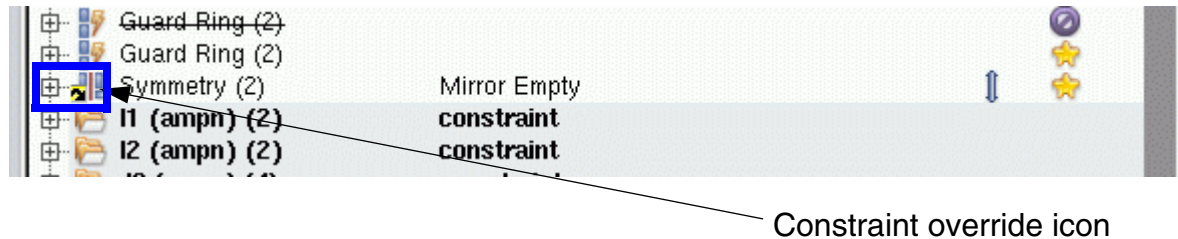


3. Select *Yes* to create an override for the current constraint in the current cell's constraint view.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

The selected constraint will now display with an override icon to the left of it.



Note: Additional override information will be displayed in the tooltip for the constraint that has been overridden.

Important

Remember to save the constraint view to store the new override settings.

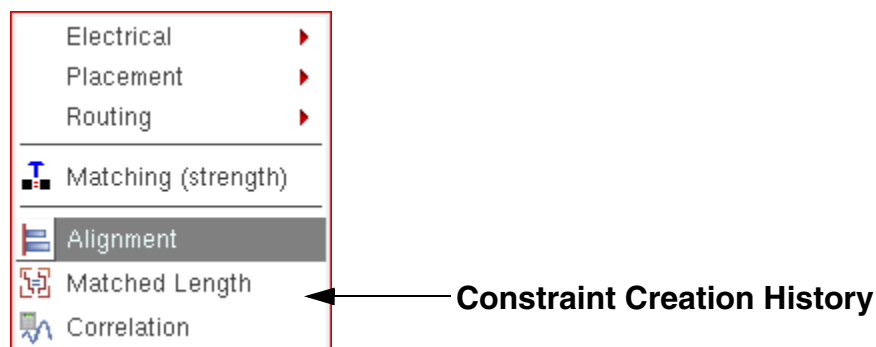
Constraint Creation History

The latest constraints to have been created in the current design are listed at the bottom of the *Constraint Creation* menu.

This list provides you with information on the latest constraint creation history and also a shortcut to creating the same type of constraint again.

Constraint history is saved and restored between sessions.

Note: Separate constraint history lists are shared across schematic *Constraint Managers* and layout *Constraint Managers*.



Setting Constraint Creation History Limits

The default constraint history list size is 5. However, this can be changed in the CIW or the `.cdsenv` file using the `constraintMenuHistorySize` environment variable. For example:

```
envSetVal ("constraint" "constraintMenuHistorySize" 'int 10)
```

Virtuoso Custom Digital Placer and Creating Constraints on Pins/Pin Figures

The Virtuoso Custom Digital Placer and the constraints SKILL functions can be used create constraints on individual pin figures (geometries).

For pins, the naming convention is: <terminalName>:<pinName>

For pin figures, the naming convention is: <terminalName>:<pinName>:<pinFigName>

Both objects will be displayed in the Constraint Manager assistant, or SKILL function output, as 'pin type objects.

For more information, see Pin Placement Constraints in the *Virtuoso Custom Digital Placer User Guide*.

Constraint Templates

A constraint template is a managed collection of constraints.

Constraint templates are pre-dominantly used to bring together a set of detailed constraints into a more abstract collection in order to manage them as one. Templates can be used to automatically manage constraints for functional groups of devices, such as analog differential pairs, cascodes, and so on.

The Constraint Manager assistant has built-in example templates, matching (strength) and Common Centroid. The former being used, depending on the strength that you set, to create more/less constraints for a device pair when the strength is set higher or lower respectively. This is performed in order to have the layout match up to a higher or lesser extent accordingly. And, the latter *Common Centroid* template being used to create a collection of constraints that implement a common centroid for two or four selected devices.

Template Management

Template management is performed using two SKILL callbacks:

1. Templates are **created** using a *create* callback that uses a list of design objects, and an optional set of arguments and parameters, to create and return the set of constraints that will then become the template members.

The *Constraint Manager* automatically creates a template and stores this returned set of constraints as template members, in effect a managed collection of constraints.

The templates also feature template parameters, which are equivalent to constraint parameters. These are defined in the template definition either through static values or SKILL expressions.

Note: The template parameters are a new capability. Therefore, the create callback creates (collects) the constraints followed by the creation of a “managed” collection.

2. Templates are **managed** using a *modify* callback, where every time a template member constraint is modified (the member or its parameters) the modify callback will be executed to validate the edits.

For more information on the template callbacks, refer to the *Virtuoso Unified Custom Constraints Configuration Guide*.

Note: This is similar to consistency checks that can be set to be performed for constraints.

The modify callback can:

- ❑ *Reject* any changes (returns `nil`) so that the constraint edit is rejected and the constraint then returns to its previous state.

Note: A modify callback that always returns `nil` will cause the constraints in a template to be read-only. A modify callback that only allows for the first constraint to be edited, may control the template to follow that of the “leader” constraint’s parameters.

- ❑ *Accept* any changes (returns `t`) so that the constraint edit is accepted and the template content is not changed (apart from deleted constraints which will be removed from the list).
- ❑ *Accept* any changes (returns a list of constraints). Here, the template content is replaced with the new constraint collection being returned.

The constraints that are now no longer managed by the template, and were not destroyed by the callback, will now become visible in the *Constraint Manager*.

Note: The modify callback is called automatically by the constraint system with the template ID and the constraint under edit as parameters. That is, the parameters are passed at the time of the call, not later.

When the callback is not a defined or callable SKILL procedure, the template content will be read-only. The modify callback of a template can *only* edit or delete constraints that are members of that template. Modifying another constraint will not re-trigger the callback (that is, the first callback has authority over the template until the callback returns).

Editing a Template

Assuming that a user-defined template has a modify callback, it is possible to:

- Edit constraints in templates (for example, change constraint parameter values)
- Add/remove members to/from constraints in templates
- Add constraints to templates
- Delete constraints from templates
- Update template parameters value

Note: When a constraint is deleted it is removed from the list of constraint templates. Whereas a constraint that has been modified is always passed as the parameters argument of the check function.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

- Dynamically add/remove/change constraints to/from the template, and the *Constraint Manager* will respond to whatever changes the callback makes.
- Similar to the constraint manager responds to constraint add/remove/change that the callback makes, it can also respond to changes made to the template parameters.

Defining a Template

New template definitions can be created using the `ciTemplateCreateDefinition` SKILL function. Using this command, you will define the template type name, the check callback, the create callback, and (optionally) whether or not the template should accept user parameters when created. Optionally, it is also possible to define a collection of template parameter definitions, (equivalent of constraint parameters for constraints).

Note: The template type name is used to look-up the template definition at a later point.

You should ensure that the template is declared and that the two callback functions (check and create), referred to by the definition, are loaded.

Important

It is recommended that you declare any template types, and load the SKILL function definitions, at startup. If a template has not been declared at initialization, a warning message will be displayed stating that the template has yet to be defined and that it should be declared.

The example below shows how to define a template and register it in the *Constraint Manager* as a constraint generator.

Example

```
;;
;; (c) Copyright 2012 Cadence Design Systems Inc. ALL RIGHT RESERVED ;;;;;;;;;
;;;;;;;;;;;;;
;; How to define a template, register it in the browser as a
;; constraint generator
;; define two callbacks: one create callback and one modify callback;
;; the create callback is the "macro" or "script" that given an ORDERED
;; set of members and a set of user parameters, creates a set of
;; constraints; the modify callback is called when any edits are
;; attempted on the member constraints of a template: edit, delete,
;; members change, etc. The modify callback receives external
;; parameters that tell which template is modified and which particular
;; constraint, user parameters or template parameters are modified

;; *****
;; NOTE: template parameters, usage, definition and interaction with template callbacks
```

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

```
;; are described and demonstrated in a dedicated section to Template Parameters for a matter of clarity.
;; *****

(defun demoPairCreateCallback (cache ;; the constraint cache where the template is created
                             listOfDevices ;; the list of design objects (same format as ciConCreate
                             @key (params nil) ;; list of template parameters described in dedicated sections
                             (userParams nil) ;; any list; not stored or interpreted by Virtuoso
                             )
  (let (newCon conList)
    ;; many checks can go in here; we skip them for clarity
    ;; this creates constraints
    newCon = (ciConCreate cache 'matchedParameters ?members listOfDevices)
    (when newCon conList = (cons newCon conList))
    (unless (car userParams) == "low"
      newCon = (ciConCreate cache 'symmetry ?members listOfDevices)
      (when newCon conList = (cons newCon conList))
      ;; two constraints for medium

    )
    (when (car userParams) == "high"
      newCon = (ciConCreate cache 'correlation ?members listOfDevices
                             ?params (list (list "coefficient" 0.8 )
                                           (list "correlatedParameters" "m w"))
                             )
      (when newCon conList = (cons newCon conList))
      ;; three constraints for high
    )
    ;; the only acceptable return value is a list of constraint user
    ;; types; these are the constraint members of the template
    conList)
  )

(defun demoPairCheckCallback (template ;; the template being modified (ci user type)
                             @key (constraints nil) ;; the constraint being modified (ci user type)
                             (params nil) ;; template parameter described and demonstrated in a dedicated section.
                             (userParams nil) ) ;; any list; (same intent as create CB)
  t ;; accepts all changes
  )

(defun demoPairCheckCallback (template ;; the template being modified (ci user type)
                             @key (constraints nil) ;; the constraints being modified (ci user type)
                             (params nil) ;; template parameters described and demonstrated in a dedicated section.
                             (userParams nil) ) ;; any list; (same intent as create CB)
  nil ;; rejects all changes, all constraints are read-only when in this template
  ;; any other value is treaded
  ;; as nil and changes are not permitted
  )

;; declare the template definition -- this is called at
;; start-up (CDS init, lib init, PDK)
(ciTemplateCreateDefinition "mypair" ;; the template type "name"
  'demoPairCheckCallback ;; check with callback
  'demoPairCreateCallback ;; create with this callback
  ?params ;; template parameter definition list described and demonstrated in a dedicated section
  ?acceptsUserParams t) ;; if nil, no user parameters passed to callbacks

;; add the constraint generator in the browser make sure you have an
;; icon that exists; this example uses the "add" icon; put your icon
;; in .cadence/dfII/icons
ciRegisterConstraintGenerator(
  list(nil
    'name "My Pair Demo"
```


Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

```
'description "Generate Various Levels of Constraints for a pair"
'expression "(demoNewPair args insts cache)"
'addToToolbar t
'iconName "add"
'args list(
    list( "strength" `enum "low" "medium" "high")
)
)

;; this get called when the UI button gets pressed
(defun demoNewPair (args insts cache )
  ;; call the template creation API
  (ciTemplateCreate cache ;; the cache storage
    "mypair" ;; the template type
    ?members insts ;; template member
    ?params ;; template parameters described and demonstrated in a dedicated section
    ?userParams (list args->strength) ;; user parameters for
    ;; creation, does not
    ;; get stored
  )
)
```

Creating a Template Instance

Template instances can be created using the `ciTemplateCreate` SKILL function. For more information on how to create a template in SKILL, refer to the last two sections of the above [Example](#).

Using this command, you need to define the constraint cache, the template definition name, the template type name, the list of template members, optionally any user-defined parameters for the template, and a collection of template parameters.

Template Parameters

Template parameters are the equivalent of constraint parameters but are applicable to templates, and also have extra features over constraint parameters.

Constraint and Template Parameters

They appear in the Parameter editor panel in the Constraint Manager and are accessible through the parameter property on template objects and are stored in the design along with the constraint template.

Template Parameters v/s Constraint Parameters

Template parameters are defined through a parameter definition using SKILL. These definitions are then passed to `ciTemplateCreateDefinition()` during the template definition creation process. Template Parameter Definitions can be defined using static values and using SKILL expressions. These expressions provide a dynamic way to define template parameters definition. In contrast to static values, these expressions are evaluated whenever a template is instantiated, making the parameter definition flexible, and dynamic.

Template Parameter Definition

Template parameters all have their own definition object which is specified through a SKILL list that features:

- A name for the template parameter
- A data type (same data types available as for constraint parameters)
- An optional default value
- An optional range of possible values

Template Parameter Definition: Format

A template parameter definition is of the following format:

```
list( <t_parameterName> <s_dataTypeSymbol> [<g_OptionaldefaultValue>]  
[<OptionalRangeOfPossibleValuesList>])
```

Where:

- `t_parameterName`: string defining the parameter name
- `s_dataTypeSymbol`: symbol that defines the data type to be used, the value of this symbol can be:
 - ❑ `string`: simple text string
 - ❑ `int`: an integer number
 - ❑ `bool`: a Boolean
 - ❑ `float`: a floating point number
 - ❑ `floatrange`: range of float of form `list(min max)` where min and max are float numbers

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

- ❑ `inrange`: range of integer of form `list(min max)` where, `min` and `max` are integer numbers.
- ❑ `stringtbl`: a list of string of form `list("string1" ... "stringN")` where, `stringN` is a string.
- ❑ `enum`: Enumeration of string tokens of form `list("token1" ... "tokenN")` where `tokenN` is a string representing a token.
- ❑ `intrange1dtbl`: list of range of integer of form `list(list(min0 max0) list(min1 max1) ... list(minN maxN))` where, `minN` and `maxN` are integers.
- ❑ `floatrange1dtbl`: list of range of integer of form `list(list(min0 max0) list(min1 max1) ... list(minN maxN))` where, `minN` and `maxN` are floats.
- ❑ `time`: datetime number.
- `g_optionaldefaultValue`: Optional symbol or list (depending on the data type) specifying a default value.
- `l_optionalRangeOfPossibleValuesList`: Optional list of possible values applicable to certain data types.

Template Parameter Definition: Usage

Template parameter definition are passed as a list of definitions as an argument `ciTemplateCreateDefinition`.

Example

In this example, `myParamsDefinitionList` is the list of template parameter definitions that defines a set of parameters and is then passed to `ciTemplateCreateDefinition()`.

```
;; strength: enumeration parameter can take values "low" "medium" "high" and its default is "low"
;; doubleNoDefault : float parameter with no default or range
;; doubleWDefault : float parameter with default value 3.75 and minimum value only of 2.5 (open range)
;; doubleWDefault2 : float parameter with default value 3.75 and minimum value range of 0.1 and maximum 4.2 (full range)
;; intNoDefault : integer parameter with no default or range
;; intWDefault : integer parameter with default value 5 and minimum value range of 1 (open range)
;; intWDefault2 : integer parameter with default value 5 and minimum value range of 1 and maximum 6 (full range)

myParamsDefinitionList = list(list("strength" 'enum "low" list("low" "medium" "high"))
  list("doubleNoDefault" 'float)
  list("doubleWDefault" 'float 3.75 2.5)
  list("doubleWDefault2" 'float 1.5 0.1 4.2)
  list("intNoDefault" 'int)
  list("intWDefault" 'int 5 1)
  list("intWDefault2" 'int 5 1 6))
```

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

```
myNewTemplateDefinition = ciTemplateCreateDefinition("matchNewCreateCB" ;; template name definition
'_ciMatchModifyCallback;; modify callback
'_ciMatchCreateCallback ;; create callback
?params myParamsDefinitionList ;; optional list of template parameter definition
?acceptsUserParams t);; optional flag that specifies if user parameter are allowed
```

Note: For more information about `ciTemplateCreateDefinition()`, refer to the *Virtuoso Unified Custom Constraints SKILL Reference*.

Template Parameter Expressions

The default value, `g_OptionaldefaultValue` and range of possible values, `(l_OptionalRangeOfPossibleValuesList)` for a parameter definition can dynamically be defined using SKILL expressions.

An expression needs to be enclosed in a double quoted string of the following form:
“<Expression>”.

<Expression> can be any SKILL expression.

For more information on SKILL expressions, refer to the SKILL documentation.

Note: An expression is often an evaluation of a function call, that is, “`myFunction()`” or can be more complex, that is, “`6 *factor(2*3 + 5)`”.

For more details, see Section Definition of a template parameter expression.

For example,

```
;; strength enumeration with a default value
;; doubleDef float with a default value expression
;; doubleOpRange float with a default value expression and a minimum value only (open range)
;; doubleFRange float with a default value expression and a minimum value and a maximum value (full range)
;; intDef integer with a default value expression
;; intOpRange integer with a default value expression and a minimum value
;; intFRange integer with a default value expression and a minimum value and a maximum value
paramDefinitionList = list(list("strength" 'enum "(enumDefault 1)" "(getEnumList)")
list("doubleNoDefault" 'float)
list("doubleDef" 'float "(getDef 'float)")
list("doubleOpRange" 'float "(getDef 'float)" "(getFloatRangeMin)")
list("doubleFRange" 'float "(getDef 'float)" "( getFloatRangeMin t)" "(getFloatRangeMax nil)")
list("intDef" 'int "(getDef 'int)")
list("intOpRange" 'int "(getDef 'int)" "(getIntRangeMin t)"
list("intFRange" 'int "(getDef 'int)" "(getIntRangeMin t)" "(getIntRangeMax nil)"))
```

Template Parameter Instances

Once a template parameter definition is specified, it is possible to instantiate a template parameter instance. As for constraint parameter instances, in the case of template parameters, an instance is a SKILL list composed of:

- a name referring to the template parameter definition name
- a value of the type specified in the template parameter definition

Template Parameter Instances: Format

The list is of the following format:

```
list(<templateDefinitionName> <value>)
```

Template Parameter Instances: Usage

Template parameter instances are mainly used in three different locations:

- As a list of instances as a list argument of `ciTemplateCreate()` to create a template instance.

```
ciTemplateCreate (
  cache                ;; Cache storage object
  "matchTemplateWithParameters" ;; Template definition name
  ?members insts      ;; instances the template will be created on
  ?params parameterInstances ;; list of parameter instances
```

- Template parameters also interact with the template callbacks through the `?params` optional argument:

- The create callback is able to access the list of template parameters during the template creation process through the `params` optional argument. The callback can then, perform any necessary processing on the parameters and respond appropriately bases on the parameter values.

```
(defun _ciMatchCreateCallback (cache listOfDevices @key (params nil) (userParams nil))
  ;; perform any processing here
  ;; ...
  ;; now return a list of constraints base on the previous processing of the different function arguments,
  ;; amongst others, params which contains the parameters list passed to ciTemplateCreate()
  newListOfConstraints)
```

- The modify callback can access parameter values through two distinct ways:
 - A list of the updated template parameters (**after** template update) through the `parameters` property on the template argument that passes the template object being updated.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

- A list of the old values of the template parameter (**before** template update) that have been updated through the `oldTemplateParams` function argument.

Through these two arguments, the modify callback has access to both values before and after the template update.

```
(defun _ciMatchModifyCallback (template @key (oldTemplateParams nil) (constraints nil) (modType nil) (userParams nil) )
  ;; Perform any necessary processing based on the value of the function arguments, amongst which
  ;; oldTemplateParams and template~>parameters
  ;; For instance Retrieve the value of the strength parameter BEFORE update (old value)
  paramList = exists(p oldTemplateParams car(p) == "strength")
  ;; ...
  ;; ... Do any further processing step (skipped for clarity) here
  ;; ...
  ;; Retrieve the value of the strength parameter AFTER update (current value).
  paramList = exists(p template~>parameters car(p) == "strength")

  ;; finally return nil to prevent change or t to allow them, here
  t
)
```

For more information and a complete code example of the template callbacks and `ciCreateTemplateDefinition`, refer to [`ciTemplateCreateDefinition\(\)`](#) of the *Virtuoso Unified Custom Constraints SKILL Reference*.

Example

This example shows in an ordered manner:

- How to specify a template parameter definitions SKILL list.
- The call to `ciTemplateCreateDefinition()`.
- How template parameter definitions are being used
- The call to `ciTemplateCreate()` and how template instances are being used.

```
;; Specification of a list of template parameter definition, from the previous example.
myParamsDefinitionList = list(list("strength" 'enum "low" list("low" "medium" "high"))
  list("doubleNoDefault" 'float)
  list("doubleWDefault" 'float 3.75 2.5)
  list("doubleWDefault2" 'float 1.5 0.1 4.2)
  list("intNoDefault" 'int)
  list("intWDefault" 'int 5 1)
  list("intWDefault2" 'int 5 1 6))

myNewTemplateDefinition = ciTemplateCreateDefinition("matchTemplateWithParameters " ;; template name definition
  '_ciMatchModifyCallback ;; modify callback function from the previous example
  '_ciMatchCreateCallback;; create callback function from the previous example
  ?params myParamsDefinitionList ;; optional list of template parameter definition
)

;; Specification of a list of template parameter instances
parameterInstances = list(
```

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

```
list("strength" "medium") ;; "strength": definition name "medium" : value chosen for this instance
list("doubleNoDefault" 5.6) ;; "doubleNoDefault": definition name 5.6 : value chosen for this instance
list("doubleWDefault" 2.4) ;; "doubleWDefault": definition name 2.4 : value chosen for this instance
list("intWDefault" 2) ;; "intWDefault": definition name 2 : value chosen for this instance
list("intWDefault2" 3)) ;; "intWDefault2": definition name 3 : value chosen for this instance
)

insts = '("M1" inst) ("M2" inst)) ;; a simple list of instances
;; Pre-requisite: have a valid CI cache object at this point: i.e. cache = ciGetCellView()

;; Call to ciTemplateCreate()
myNewTemplate = ciTemplateCreate (
    cache                ;; Cache storage object
    "matchTemplateWithParameters" ;; Template definition name
    ?members insts      ;; instances the template will be created on
    ?params parameterInstances;; list of parameter instances is passed here
)
```

For more information on [`ciTemplateCreate\(\)`](#), refer to the *Virtuoso Unified Custom Constraints SKILL Reference*.

Creating a Template in Constraint Manager

For information on creating user-defined templates in the *Constraint Manager* see [Creating User-Defined Constraints](#) in the *Virtuoso Unified Custom Constraints Configuration Guide*.

Note: Make sure that the constraint generator is installed and an icon is associated with the template name type.

Defining the Interface for Editing a Template and its Parameters

A template and its parameters can be edited within one of the following:

- *Constraint Parameter Editor* pane of the Constraint Manager assistant
- Associated constraint generator dialog box
- Defined external constraint editor window (such as the *Module Generator Editor*)

By default, you can edit the template parameters in the *Constraint Parameter Editor* pane of the Constraint Manager assistant. However, to choose a different mode for editing, you can use the following arguments:

- The 'useCGenForEdit argument of the `ciRegisterConstraintGenerator` function
- The 'templateTypes and 'useForTemplateEdit arguments of the `ciRegisterConstraintEditor` function

Opening the Constraint Generator Dialog Box for Editing

When you set the `'useCGenForEdit` argument of the `ciRegisterConstraintGenerator` function to `t`, you can open the constraint generator dialog box in the following two scenarios:

- When you double-click the template in the *Constraint Browser*.
- When you click any template parameter for editing in the *Constraint Parameter Editor* pane of the Constraint Manager assistant.

Note: By default, this argument is set to `t` only for the generic module generators (*GenericModgen* constraints). Therefore, no other generators get affected and it is still possible to edit their parameters within the *Constraint Parameter Editor* pane.

Opening an External Constraint Editor for Editing

By default, you can open an external constraint editor only when you double-click a constraint in the *Constraint Browser*. This is managed based on the value specified with the `'constraintType` argument of the `ciRegisterConstraintEditor` SKILL function. To open an external constraint editor when a template is double-clicked in the *Constraint Browser* or a template parameter is clicked in the *Constraint Parameter Editor* pane, do the following:

1. Reset the `'useCGenForEdit` argument of the `ciRegisterConstraintGenerator` function to `nil`, which is otherwise set to `t` by default.
2. With the `'templateTypes` argument of the `ciRegisterConstraintEditor` function, specify a list of template types that can be edited in a particular constraint editor.
3. Set the `'useForTemplateEdit` argument of the `ciRegisterConstraintEditor` function to `t`, which is otherwise set to `nil` by default.

For example, the following definition of the `ciRegisterConstraintEditor` function ensures that *Module Generator* opens when templates of types *CurrentMirror* and *SeriesResistorArray* are double-clicked:

```
ciRegisterConstraintEditor(  
  list(nil 'name "Module Generator..."  
    'description "Modgen Editor "  
    'constraintType "modgen"  
    'constraintParams nil  
    'editorAvailableExpression "isCallable('mgCreateOrEdit) ")
```


Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

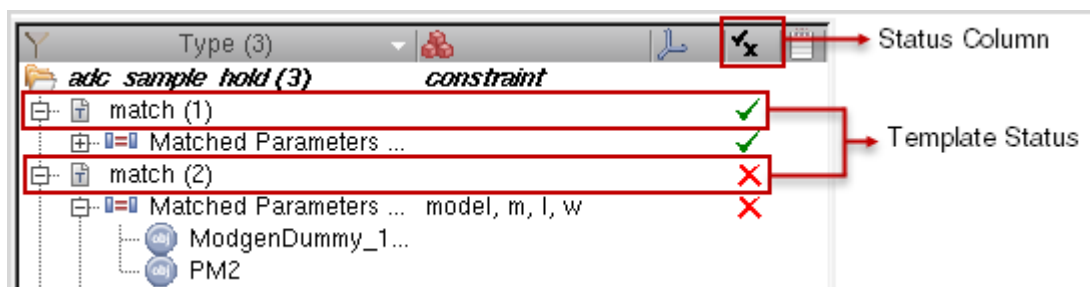
```
'startEditorExpression " mgCreateOrEdit (geGetEditCellView() constraint
nil) "
'iconName " nexGenEd_dot_dot_dot "
'addToToolbar t
'templateTypes '("CurrentMirror" "SeriesResistorArray")
'useForTemplateEdit t
)
)
```

Note:

- ❑ In the *Constraint Browser*, when you select a template that does not match the template types listed with the 'templateTypes argument, the *Constraint Editor* menu in the *Constraint Manager* toolbar appears as disabled, that is, grayed out.
- ❑ The 'useForTemplateEdit argument works only when the 'useCGenForEdit argument of the `ciRegisterConstraintGenerator` SKILL function is not set for the constraint generator associated with the concerned template. Otherwise, double-click or parameter edit launches the associated constraint generator dialog box.

Viewing Template Status in Constraint Manager

By default, the template status is not checked. It is only when you click the *Check* menu item on the *Constraint Manager Toolbar* that each template and its member constraints are evaluated and accordingly, certain icons depicting the identified status are displayed under the *Status* column, as shown below.



Note: You can click the *Status* column header to sort based on the constraint status.

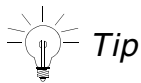
During the evaluation, the templates are checked using the associated preconditions set using the `ciRegisterConstraintGenerator` SKILL function.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

The following table lists the different scenarios in which a particular template status is displayed:

Scenario	Template Status
The specified precondition failed	✗
The specified precondition passed, but any member constraint(s) failed	✗
The specified precondition passed, but any member constraint(s) became inactive	⚙
The specified precondition passed and all member constraints passed too	✓
The specified precondition passed, but any member constraint(s) was not checked	✓



Tip

In the *Constraint Browser*, when you move the pointer on a template status displayed under the *Status* column, a tooltip is displayed. This tooltip describes the differences related to any or all of the constraint parameters, member parameters, member order, or axis parameters.

To know more about the different template status icons that might be displayed, refer to the table in the [Constraint Status](#) section.

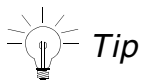
Deleting Constraints

To delete a constraint:

1. Select one or more constraints from the *Constraint Browser*.
2. Either:
 - ☐ Right click in the *Constraint Browser* and select *Delete* from [Selection and Cross-Selection of Constraints](#).
 - ☐ Select the *Delete* button from [Constraint Manager Toolbar](#).
 - ☐ Press the `Ctrl+X` keys.
 - ☐ Press the `Delete` key.

Saving and Storing Constraints (Schematics Only)

You can save those constraints detailed in the *Constraint Manager* by selecting the *Save Constraints* option on Constraint Manager Toolbar (schematics only).



Alternatively, you can use the *Save*, *Save a Copy*, and *Discard Edits* options available in the *File* menu of the Virtuoso Schematic Editor XL to handle any updates done for a constraint. When you update a constraint, but have not saved it yet, save menu options are labeled as *Save (Constraints)* and *Discard Edits (Constraints)*. See also Switching On/Off Constraint Menu Options in Schematics XL File Menu.

Note: The Constraint View can be copied, renamed, or deleted using the Library Manager. For more information, see the Cadence Library Manager User Guide.

Front end constraints are stored in a separate view. This *constraints view* cannot be opened directly, rather it can only be opened through the opening of a schematic or configuration hierarchy.

Back end constraints are stored in the layout cellview. The reasons for storage differentiation between layout and schematics is down to the nature of the front end, for example:

- In many flows schematics are read-only. Constraint editing should still be allowed when schematic views are read-only.
- Support of a hierarchy and configuration requires that constraints refer to an entire design rather than just the current view.
- Future support of a text view is accounted for by keeping constraints separate from the design view types.

Switching On/Off Constraint Menu Options in Schematics XL File Menu

You can use the comanageConstraintView environment variable to switch on/off the display of the *File – Save (Constraint)* and *Discard Edits (Constraint)* menu options in Schematics XL.

Note: Switching off this option can speed up performance because the constraint infrastructure will no longer be queried to find the associated constraint view to update the status of the schematic *File* menu items for constraints.

Selection and Cross-Selection of Constraints

Whenever one or more constraints are selected in the *Constraint Browser*, Constraint Parameter Editor will update to present the parameters and parameter values of the selected constraints.

Note: If you select multiple constraints from the *Constraint Browser*, you can edit multiple constraints' parameters at the same time. For more information see Editing Constraint Parameters.

Any constraint, or member object, selections that are made in the *Constraint Manager* will be reflected in the design canvas. In such circumstances, the canvas will auto-pan to the selected constraint or member object, helping to visualize the constraint. The *Constraint Manager* also interacts directly with the Navigator and the Property Editor. That is, any selections made in the *Constraint Browser* will be reflected in both of these assistants. In addition, the *Constraint Manager* reacts to selection/deselection notifications from the Search assistant and the design canvas. Notifications can be sent out using hsmSelect.

Note: Standard selection utilizes `Shift+Select`, `Ctrl+Select`, and `deselect all`.

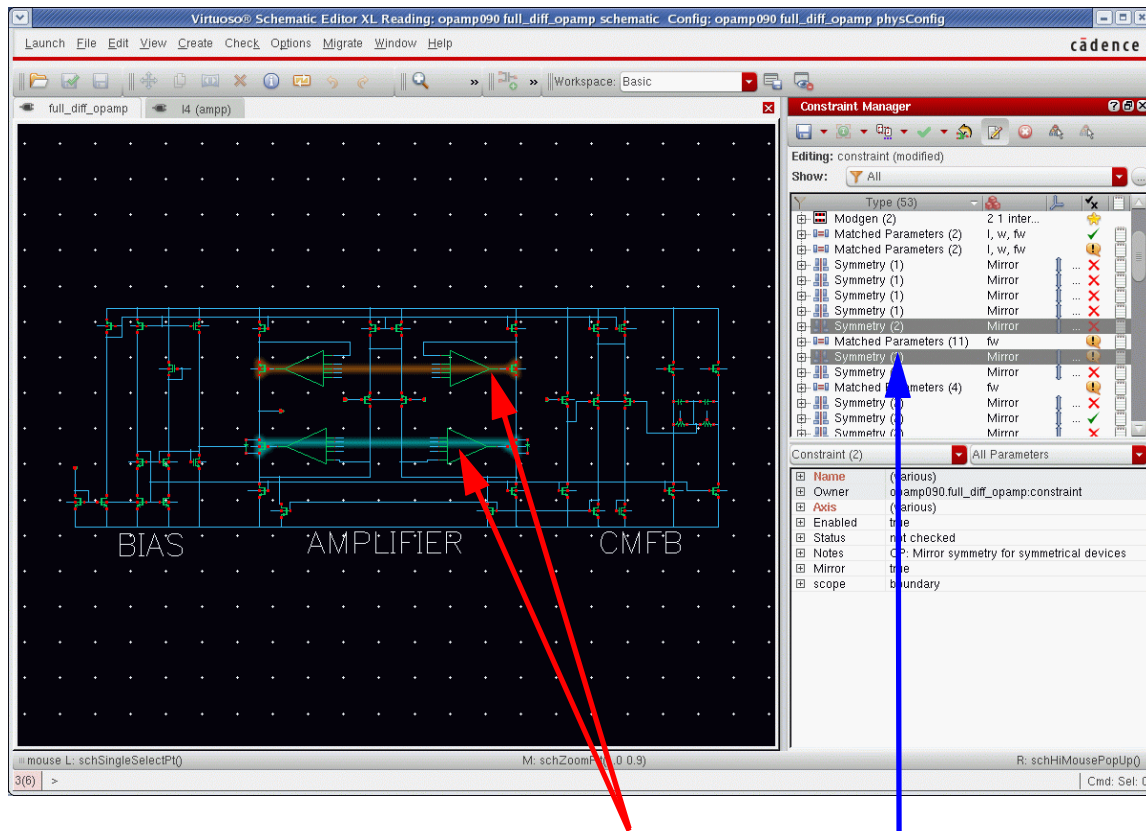
Halo Highlighting

If you have halo highlighting *on*, by ensuring that *Halo* has been selected from the Constraint Browser Context-Menus, any selections made in the *Constraint Manager* will cause the constraint members to be haloed/highlighted on the canvas.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Note: Constraint haloing is fully supported for lower-level cell constraints (see also [Creating Constraints at Different Design Levels](#) and [Cross-Selection of Constraints Between Schematic and Layout Constraint Managers](#)).

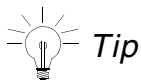


Two symmetry constraints selected in the Constraint browser and highlighted in the design canvas.

Selecting a constraint will cause halos to be drawn around each constraint member on the canvas, with a connecting line being drawn between all objects that belong to the currently selected constraint.

If you have selected multiple constraints in the *Constraint Manager*, this will result in multiple constraints being haloed in the canvas simultaneously, with each constraint type being haloed in a different color.

Note: You cannot currently plot halos. You can however print the constraint details (see the [Custom Constraints Functions](#) chapter in the *Virtuoso Unified Custom Constraints SKILL Reference*). It is also possible to export canvas details, using the *File – Export Image* option or using `hiExportImage`.



Tip

When a constraint is selected in the *Constraint Manager*, use the *View – Display Dimming* option in VSE to dim all objects in the design canvas to better highlight the location of the selected constraint. For more information, see [Display Dimming](#) in the *Virtuoso Schematic Editor User Guide*.

See also:

- [ciSetHaloPolicy](#) and [ciSetMaxHaloGroupSize](#) for information on how you can set halo policy and threshold values for halo group sizes.

Cross-Selection of Constraints Between Schematic and Layout Constraint Managers

Cross-selection of constraints between the schematic *Constraint Manager* and the layout *Constraint Manager* is controlled by the [crossHalo](#) environment variable, as defined in the constraint partition.

Cross-Selection at Different Design Levels

To better understand cross-selection at different levels of a design, consider the following scenario:

- Firstly, there is a hierarchical schematic design where there are constraints at each level in the hierarchy. The schematic *Constraint Manager* will display these constraints in a hierarchical manner.
- Secondly, a layout for the top design where an *Update Layout Constraints* action is performed in the physical *Constraint Manager*.
- A hierarchical constraint transfer now occurs (that is, the top-level layout will now have all the constraints that were present in the entire logical hierarchy). This will happen irrespective of whether the layout is flat or hierarchical.
 - If the layout is flat, all the constraints would typically be in-context, otherwise they will be out-of-context and would not be visible in the physical (layout) *Constraint Manager*.
 - If the layout is hierarchical, the lower-level layouts could also have constraints, but they are there because the *Update Layout Constraints* option might have run in to those lower level layouts.

Taking the above scenario into consideration, cross-selection of constraints at different levels of a design works as follows:

Given a schematic *Constraint Manager* that is displaying the constraints in the hierarchy of a design called “top”, and a lower-level cell is now selected in the schematic *Constraint Manager*, then the constraints cross-selection should occur only in the physical *Constraint Managers* that are currently displaying the constraints of those designs that comprise a valid (logicalCV physicalCV) VLS XL pair.

No attempt is made to cross-select constraints in a physical *Constraint Manager* that might map to the lower level design (that contains the logical constraint) on the SRC side. Similarly, if a constraint is selected in the physical *Constraint Manager*, cross-selection should only happen in the logical *Constraint Manager* that is associated with a design that comprises a (logicalCV physicalCV) VLS XL pair with the design with which the physical *Constraint Manager* is associated. Inside the logical *Constraint Manager*, only the constraint that has generated the physical constraint should be cross-selected.

Configuring Constraint Halo Display

Constraint visualization functionality, provided in the Set Constraint Halo Options form, offers designers the capability to set particular halo colors, and their style attributes, to one or more chosen constraints.

To access and configure halo display:

1. Right-click over the *Constraint Browser* to display the *Constraint Browser* context-menu.
2. Select *Options* to display the Constraint Manager Options form.
3. Ensure that *Enable* is checked in the *Halos* section (it is checked by default).
4. Click the *Setup* option to display the Set Constraint Halo Options form.

Note: Alternatively, you can open this form by running the [ciSetHaloOptions](#) SKILL function in the CIW.

The Set Constraint Halo Options form comprises of two tabs:

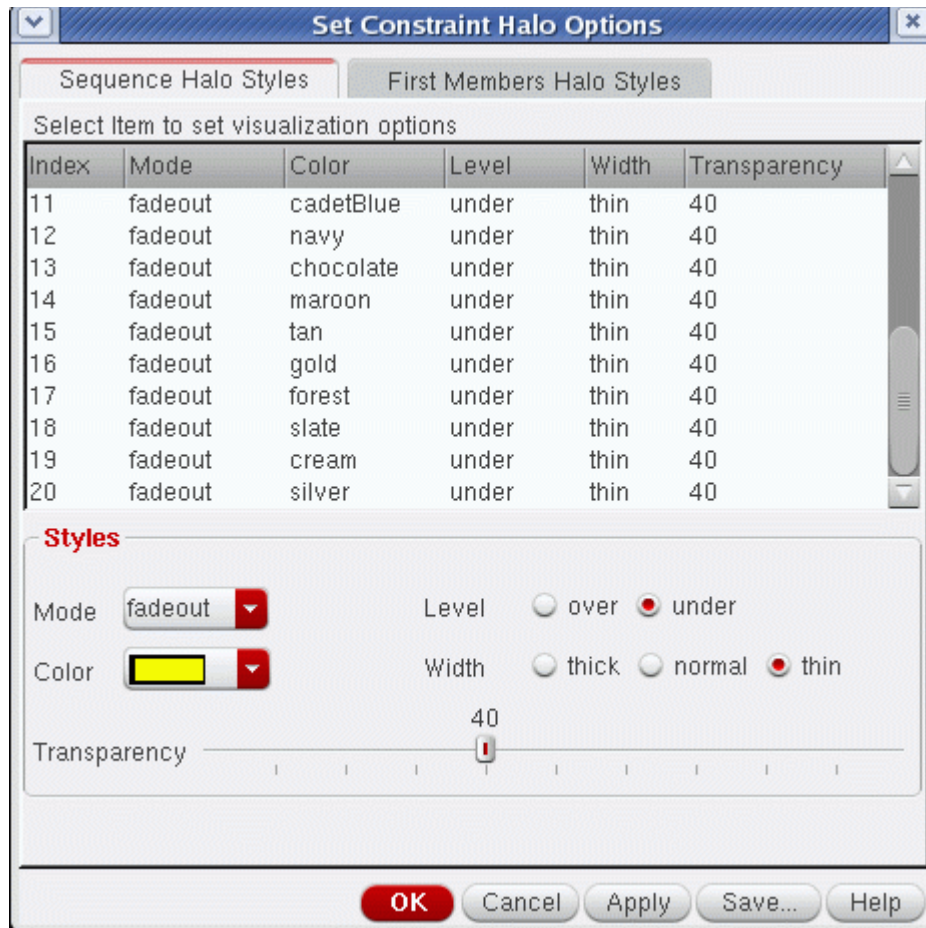
- The [Sequence Halo Styles Tab](#) that lists the current constraint color *Index* items.
- The [First Members Halo Styles Tab](#) that is used to apply the constraint style to one or more selected constraint types.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Sequence Halo Styles Tab

The Sequence Halo Styles tab contains a library of (20) default constraint halo visualization *Index* items that can be edited using the halo *Styles* options.



The *Sequence Halo Styles* tab contains the following options:

Select item to set visualization options

Select the constraint halo color item (1 to 20) that you want to edit.

The *Index* rows list in which order a halo style will be applied. For example, if only 3 constraints are selected, the first 3 style combinations will be used for halos on each member group in the design canvas.

Styles

Use the following display options to control the visualization of the selected constraint *Index* item.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

- *Mode*

Select the constraint highlight mode (from *fadeout*, *glow*, *plain*, *random*, and *none*). By default, *fadeout* is the highlight mode which displays halos.

Selecting *glow* will also display the halo effect, while *plain* mode displays without haloing (*plain Mode* however supports alpha blending and underlay/overlay modes).

Choosing *random* will display constraint highlighting in scattered dots.
- *Color*

Select the color of the constraint halo *Index* item.
- *Level*

Select the location of the halo style from either *over* or *under* (the default).
- *Width*

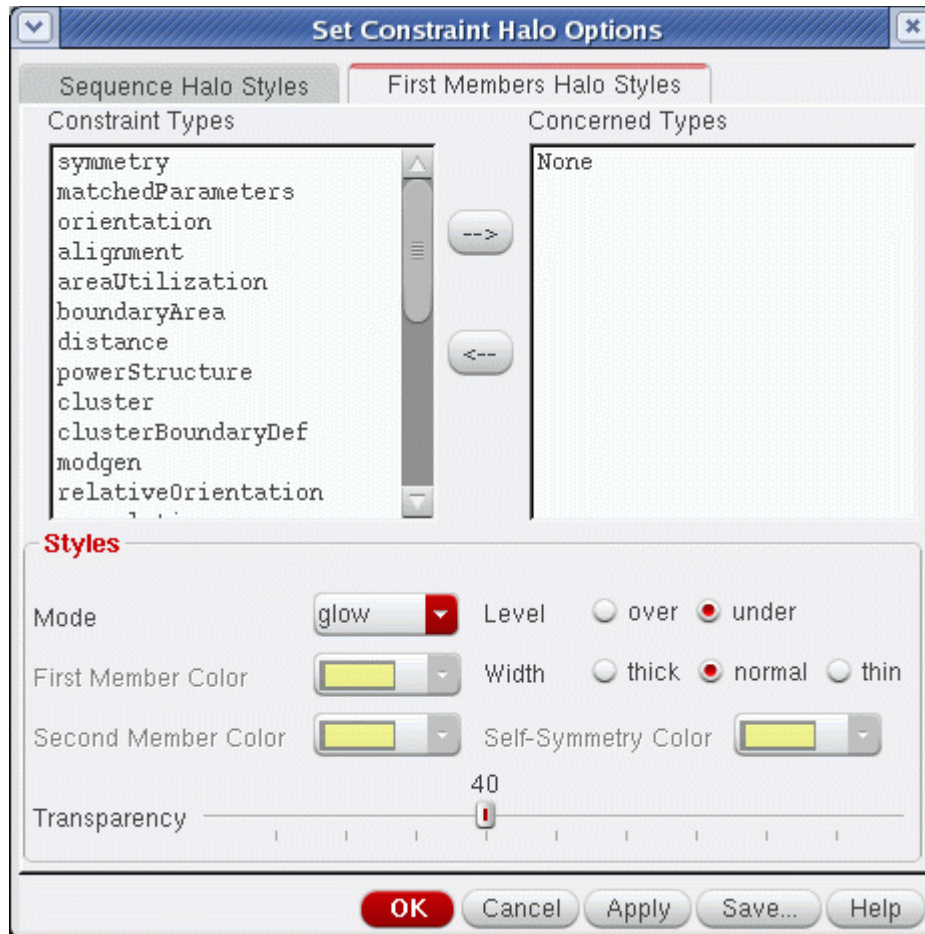
Select the width of the constraint highlight lines.

You can choose from *thick*, *normal*, or *thin*. The default being *thin*.
- *Transparency*

Select the transparency setting from a range of 0% - 100%, the default being 40%.

First Members Halo Styles Tab

The First Members Halo Styles tab contains a list of *Constraint Types* that you can select to apply a particular halo style to.



For some constraint types the first member has a particular role, for example it could be a reference for the other members. It may therefore be valuable to highlight it in a particular manner, for example by using a larger width for its halo style.

The *First Members Halo Styles* tab contains the following options:

Constraint Types

Lists the constraint types available to be selected and assigned a particular color scheme (by applying the options in the *Styles* section).

Concerned Types

Lists the constraint types currently selected to have a particular color scheme applied.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Styles

Use the following options to control the visualization of the selected *Concerned Types*.

- Mode

Select the constraint highlight mode (from *fadeout*, *glow*, *plain*, *random*, and *none*). By default, *fadeout* is the highlight mode which displays halos.

Selecting *glow* will also display the halo effect, while *plain* mode displays without haloing (*plain Mode* however supports alpha blending and underlay/overlay modes).

Choosing *random* will display constraint highlighting in scattered dots.

- Level

Select the position location of the halo style from either *over* or *under* (the default).

- First Member Color

Note: This option is only active for the *symmetry* constraint.

Select a unique color for the first *symmetry* constraint member. For a *symmetry* constraint this would visualize, in the schematic, which member should be placed on the left side of the symmetry axis in the layout.

For *symmetry* constraints that have two members, all first members will be haloed and grouped together. This provides traversal, rather than linear groupings, for each constraint.

- Width

Select the width of the constraint highlight lines.

You can choose from *thick*, *normal*, or *thin*. The default being *thin*.

- Second Member Color

Note: This option is only active for the *symmetry* constraint.

Select a unique color for the second *symmetry* constraint member. For the *symmetry* constraint this would visualize, in the schematic, which member should be placed on the right side of the symmetry axis in the layout. For *symmetry* constraints that have two members, all second members will be haloed and grouped together. This provides traversal, rather than linear groupings, for each constraint.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

- *Self-Symmetry Color*

Note: This option is only active for the *symmetry* constraint.

Select a color for any self-symmetry constraint members (this will be a unique member of all *symmetry* constraints with one member only). For those *symmetry* constraints with only one member currently selected in the *Constraint Manager*, all members will be haloed and grouped together in the same selected color.

- *Transparency*

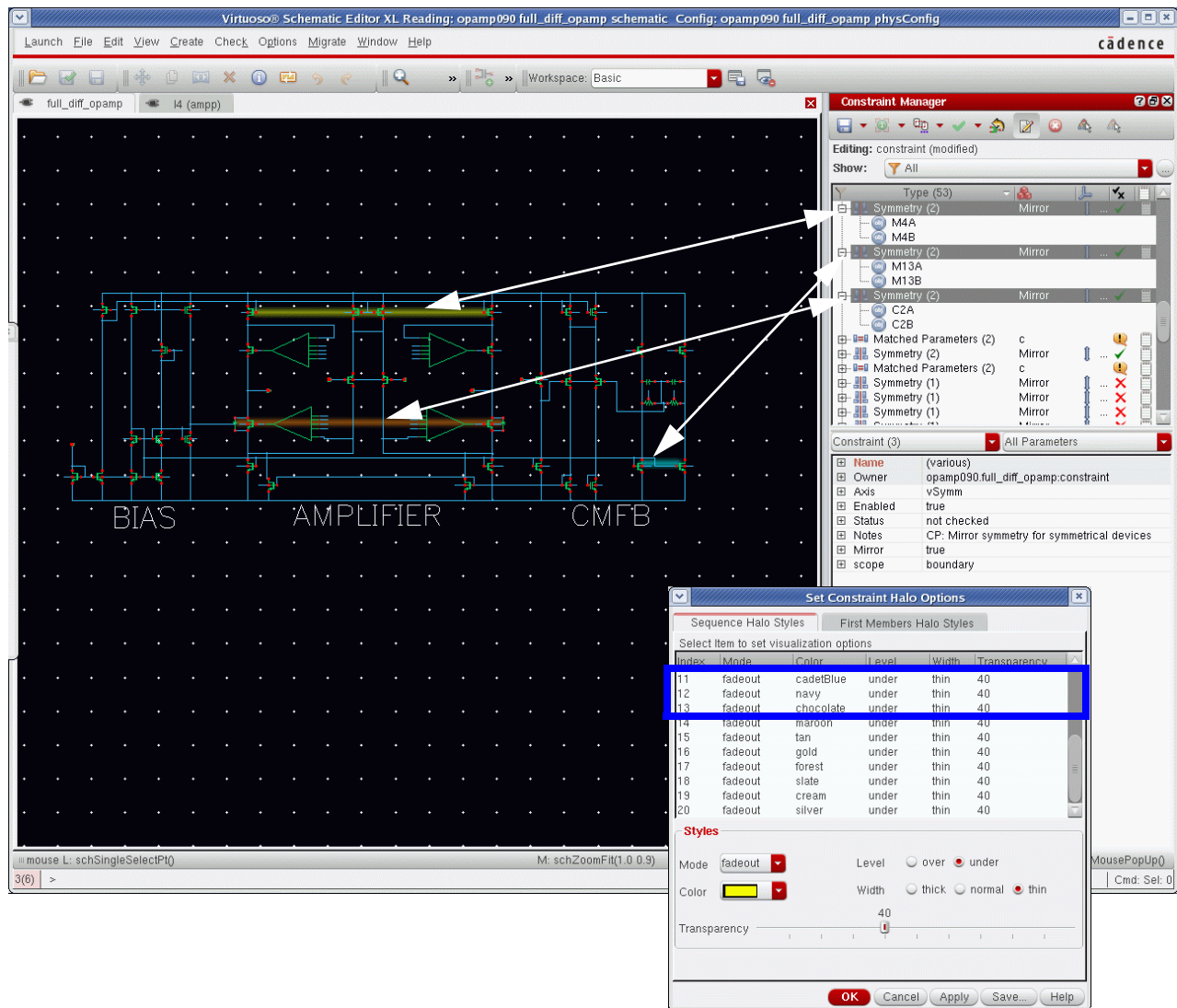
Select the transparency setting from a range of 0% - 100%, the default being 40%.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Halo Index Styles Examples

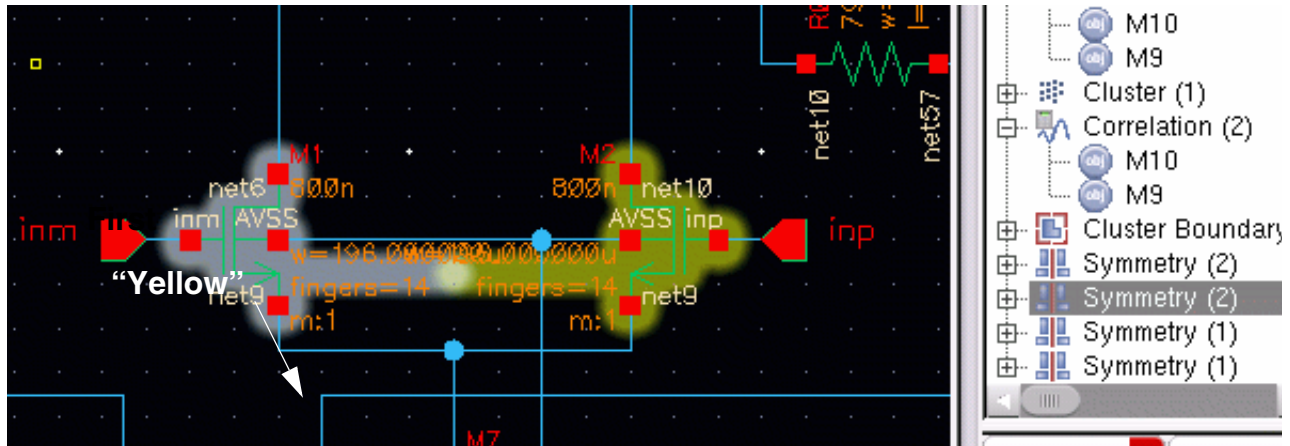
- In the screenshot below, three symmetry constraints have been selected in *Constraint Manager*, with each constraint having a different *Index* style applied (colors yellow, cyan, and orange used to distinguish different constraints).



Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

- In the screenshot below, a symmetry constraint is shown with different colors applied to distinguish the first (in yellow) and the second (in white) constraint member.



Saving Constraint Halo Options

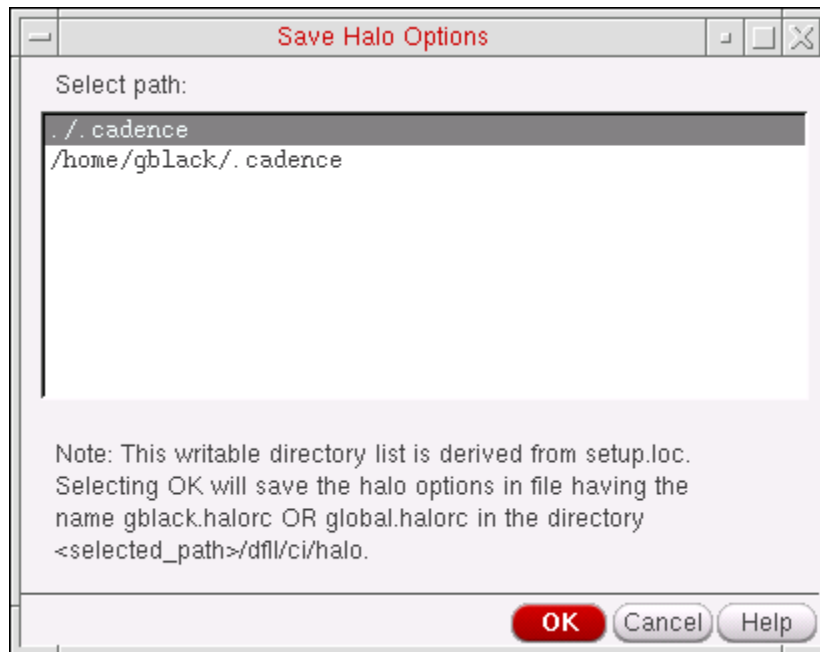
If you click the *Save* button in the Set Constraint Halo Options form this will display the Save Halo Options form.

In the Save Halo Options form, select the path location (your home directory for local use or the `.cadence` directory for site-wide use) where you want to store the customized halo options and they will be available for use across various sessions.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Note: Constraint halo options are read from the file that is retrieved using the search filter criteria that has been specified in `setup.loc`. Therefore, there could be site-specific, user-specific or design-specific halo option files.



Adding Constraint Members

You can add and remove constraint members at any time, as well as edit constraint parameters, notes, axes, and so on.

To add a new constraint member to a constraint:

1. Select in the canvas or from the *Navigator* assistant the objects that you want to add as members to a constraint.
2. In the *Constraint Manager*, right-click the constraint to which you want to add members.
3. Choose *Add Member* from the Constraint Browser Table Context-Sensitive Menu.

The selected constraint's membership expands to include the selected objects.

Note: Some constraint types have restrictions on the number of members that they can contain and also on the type of members that can be mixed within a constraint. For example, it might not be possible to mix instances and nets in the same constraint. If an anomaly is identified, a warning message is displayed stating that the constraint cannot be created. For information about members that can be mixed within a constraint, see Default Constraint Types.

Deleting Constraint Members

To delete a constraint member:

1. Select one or more constraint members (using the `Shift` or `Control` keys) from the *Constraint Browser*.
2. Either:
 - ☐ Select the *Delete* button from Constraint Manager Toolbar.
 - ☐ Press the `Delete` key.

Note: You cannot delete a constraint member (or members) if the constraint would be left with less than the minimum required members (objects) for that particular constraint type. For information on deleting the whole constraint see Deleting Constraints.

Interactive Constraint Searching

You can use interactive *Search* functionality to help locate design components. This can be particularly useful when trying to populate constraints or templates with nets and instances that are located in large, complex designs.

In schematics, the *Search* assistant searches for constraints throughout the entire design hierarchy, starting at the current edit cellview (rather than the top cellview scheme as followed by the *Navigator*) and then displays the lower level design constraints. Selecting these constraints causes the *Search* assistant to send constraint selection notifications to the *Constraint Manager* (via hierarchical selection management (HSM)), that is hierarchical in nature which the *Constraint Manager* can successfully handle.

Note: Cross-selection will only work with instances, instance terminals, nets and pins.

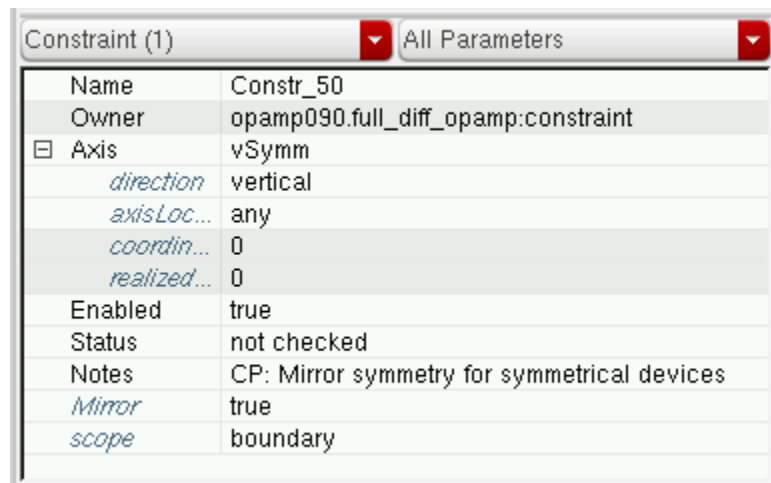
For more information on interactive search, see [The Search Assistant and Toolbar](#) in the [Virtuoso Schematic Editor User Guide](#).

Constraint Parameter Editor

The lower section of the *Constraint Manager* has an integrated *Constraint Parameter Editor*. You can use this editor to add and/or edit parameters and parameter values for one or more constraints, or constraint members, that have been selected in the *Constraint Browser* as well as view the [Constraint History](#).

The *Constraint Parameter Editor* pane can be displayed by:

1. Ensuring that the *Constraint Parameter Editor* option is selected in the *Constraint Manager* toolbar.
2. Double-clicking a constraint in the *Constraint Browser*.



The screenshot shows the 'Constraint Parameter Editor' window. At the top, there are two dropdown menus: 'Constraint (1)' and 'All Parameters'. Below these is a table with the following data:

Name	Constr_50
Owner	opamp090.full_diff_opamp:constraint
☒ Axis	vSymm
<i>direction</i>	vertical
<i>axisLoc...</i>	any
<i>coordin...</i>	0
<i>realized...</i>	0
Enabled	true
Status	not checked
Notes	CP: Mirror symmetry for symmetrical devices
<i>Mirror</i>	true
<i>scope</i>	boundary

If you right-click the *Constraint Parameter Editor* pane, a context-menu is displayed allowing you to view the table information by parameter (*View by Parameter*) or by constraint (*View by Constraint*). There are also shortcuts to expand and collapse all table contents, as well as options to scroll back and forward through the listed parameters/constraints.

Note: In the parameter filter drop-down list box, member parameters are displayed only if the member of the selected constraint has such parameters.

Selecting and Displaying Constraints and Constraint Members in the Constraint Parameter Editor

Constraints are edited by initially selecting the constraint/member that you want to edit from the *Constraint Browser* and then modifying the parameter or parameter values in the *Constraint Parameter Editor* pane. A number of constraint parameters can be edited simultaneously by making multiple selections in the *Constraint Browser*.

You must select each constraint member individually in the *Constraint Browser* for them to be displayed in the *Constraint Parameter Editor* pane. Selecting one or more constraints does not automatically select the associated constraint members.

Note: Using `Shift`-select, when constraints are expanded, will select the constraint members. However, when the constraints are collapsed the members will not be selected. Members must therefore be on view for a `Shift`-select to include them in a selection.

At the top of the *Constraint Parameter Editor* pane, you will see the following two drop-down menus:



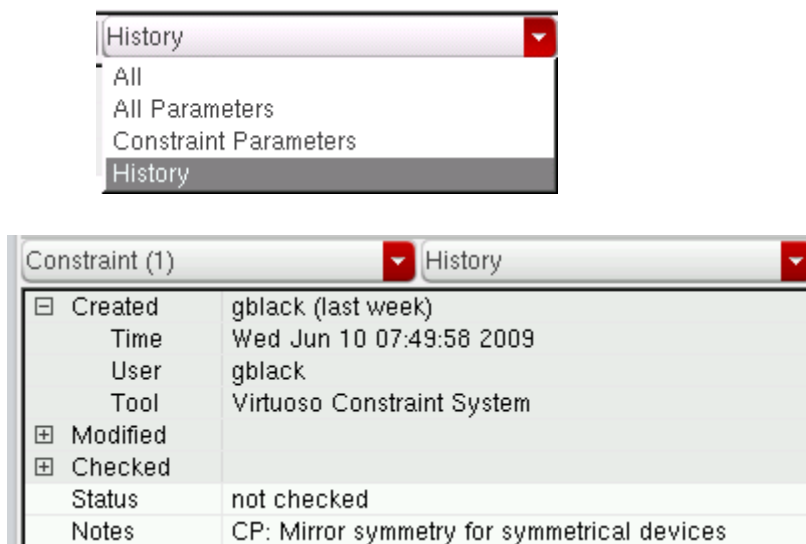
- The *Constraint Filter* drop-down menu to the left provides information on the number of constraints (*Constraints*) and constraint members (*Members*) currently selected in the *Constraint Manager*, along with a total amount (*All*). You can choose whether you want to display *Constraints*, *Members*, or *All* in the *Constraint Parameter Editor* pane.
- The *Parameter Filter* drop-down menu to the right lets you choose to view either *All* constraint information (parameters and history), only *All Parameters* information, only *Constraint Parameters* information, or only constraint type *History* for the selection made in the *Constraint Filter* drop-down menu.

See also: [Constraint History](#), [Editing Constraint Parameters](#), and [Constraint Parameter Editor Status Visualization](#).

Note: For information on all the parameters associated with each constraint type see [Default Constraint Types](#).

Constraint History

Selecting *History* (or *All*) from the *Parameter Filter* pull-down displays the creation, modification, and check attribute values for the current constraint.



In addition to related *Status* and *Notes* parameters, each constraint can have the following specific *History* parameters (also shown as constraint tooltip information):

- *Created* provides details of *Time/User/Tool* information related to when a constraint was created.

The *Tool* information indicates on which application the constraint was created, for example schematic *Constraint Manager* or layout *Constraint Manager*.

Note: Any constraints created prior to the introduction of constraint history functionality (IC6.1.2) will only display *Created* information (which reflects when they were first displayed with history information being available). The *Modified* and *Checked* values will be blank.

- *Modified* provides details of *Time/User/Tool* information related to when a constraint was last modified. For example, if a constraint member was added or removed, *Notes* added, and so on.

The *Tool* information indicates on which application the constraint was last modified, for example schematic *Constraint Manager* or layout *Constraint Manager*.

- *Checked* provides details of *Time/User/Tool* information related to when a constraint was last checked. For example, if the constraint *Status* was changed.

The *Tool* information indicates on which application the constraint was last checked, for example schematic *Constraint Manager* or layout *Constraint Manager*.

Note: In addition to being able to use the default constraint history filters, you can also set your own filters based on constraint history. For more information, see [Constraint Filters](#).

Editing Constraint Parameters

A designer can set or modify a constraint, and the member parameters of constraints in a design, using the *Constraint Parameter Editor* pane. Constraint parameters apply to the whole constraint or all of its members, while a member parameter only applies to a particular member. Member parameters, when set, will take precedence over constraint parameters.

Multiple selections in the *Constraint Browser* will cause the attributes to be rolled up and collapsed in the *Constraint Parameter Editor* pane. To view the *Name*, *Axis*, and so on, parameters of all selected constraints you have to click the “+” icon to the left of the individual parameter name. This will expand the selected constraint parameter. You can also right-click over the *Constraint Parameter Editor* and select *Expand All* from the *Constraint Parameter Editor* context-menu to expand the contents of all the collapsed parameters.

Some of the common constraints parameters found in most constraint types are: *Name*, *Axis* (*Symmetry*, *Alignment*, and *Distance* constraints only), *Enabled*, and *Notes*.

Note: In the layout view, for most constraint types, a *Status* parameter is also displayed in the *Constraint Parameter Editor* pane.

Information on how to edit these constraint parameters follow on in this section. All other parameters will follow a similar editing practice.

Note: For information on controlling what parameter names are displayed, see [Customizing Constraint Types Using a Configuration File](#) in the *Virtuoso Unified Custom Constraints Configuration Guide*.

Editing the Constraint Name Parameter

The name of a constraint is automatically generated when the constraint is first created. The default is to create the first constraint in a session as *Constr_0*, the second constraint as *Constr_1*, and so on.

Note: The name of a constraint is also displayed when the mouse cursor is moved over the constraint type in the *Constraint Browser*.

To edit a constraint's *Name* parameter:

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

1. In the *Constraint Browser*, select the constraint whose name you want to edit.
2. Click in the *Name* parameter value field in the *Constraint Parameter Editor* pane.
3. Edit the current text entry.

Note: A check will be performed to ensure that the new name is unique.

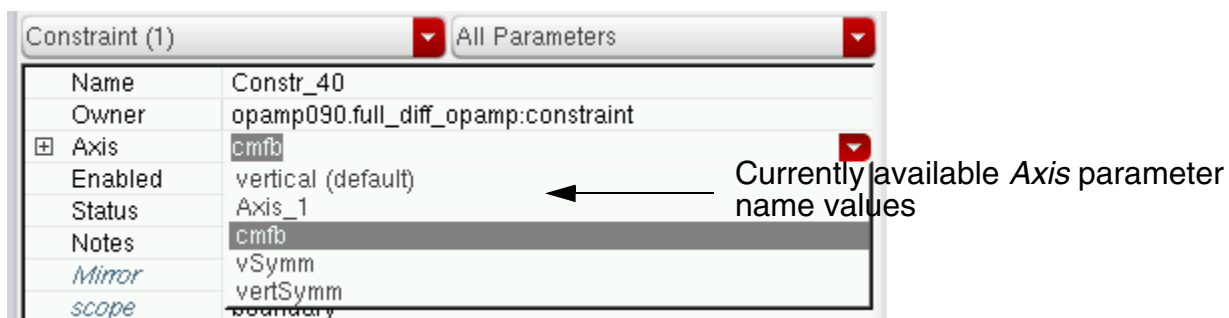
Editing the Constraint Axis Parameter

Note: The *Axis* parameter is only displayed for *Symmetry*, *Alignment* and *Distance* constraints (see also [Constraint Axis](#)).

When, for example, a *Symmetry* constraint is created using the *Constraint Manager*, it will have an axis parameter, but the field value in the *Axis* column will be left empty until you manually enter or select an existing axis name value.

To add or edit an *Axis* **parameter** name value:

1. In the *Constraint Manager* select the constraint whose axis parameter name value you want to add/edit.
2. In the *Constraint Parameter Editor* pane, either:
 - a. Click the drop-down list to the right of the *Axis* parameter and select the name value to be applied to the *Axis* parameter.or
 - b. Manually type a new axis name value into the *Axis* parameter name field (which will allow for a new axis name to be defined for the current constraint).



Note: If you require a number of constraints to share the same axis, ensure that each constraint has the same axis name value selected.

Virtuoso Unified Custom Constraints User Guide

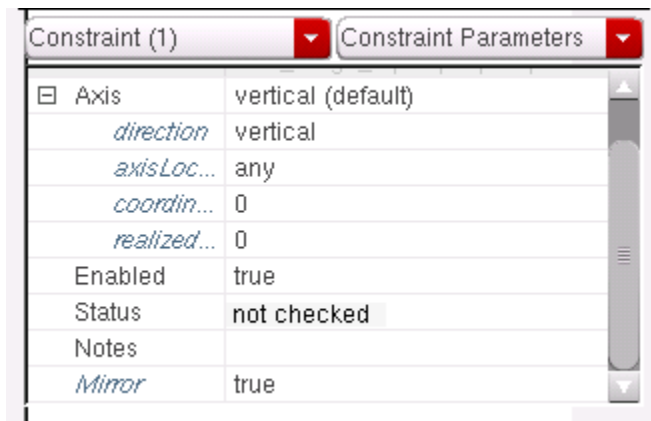
The Constraint Manager Assistant

If you do not enter a specific axis name, directly into the *Axis* parameter name field, an auto-generated axis name will be applied, for example `Axis_0`. This axis name will now also be made available from the axis parameter name pull-down for any future selections.

Editing Axis Sub-Parameters

To add/edit a constraint's axis **sub-parameter** values:

1. In the *Constraint Manager* select the constraint whose axis sub-parameter value(s) you want to add/edit.
2. In the *Constraint Parameter Editor* pane, select the *Axis* constraint parameter.
3. Click the “+” icon to the left of the *Axis* parameter to display all of the sub-parameters associated with constraints axes: *direction*, *axisLocation*, *coordinate*, and *realizedCoordinate*.



4. Edit the sub-parameter values as required using the available pull-downs and edit fields.

Any sub-parameters that you have changed from their default values will now be displayed in standard, black font (as opposed to *italic* blue font). For more information, see [Constraint Parameter Editor Status Visualization](#).

After applying the sub-parameter edits, the new axis sub-parameter values are added to the *Axis* parameter value fields.

The axis parameter edits made, for example axis name and *direction*, are also displayed in the *Axis* column of the *Constraint Browser*.

Editing the Enabled Parameter

Constraints can be enabled or disabled. Enabled constraints are to be considered in the design process while disabled constraints should be ignored.

Constraints can be disabled by setting the associated *Enabled* parameter to *false*. When a constraint is disabled, the constraint gets struck-out, in purple text, in the *Constraint Browser*.

However, if you have changed a constraint's *Enabled* parameter to be *false*, but then edit another of the constraint's parameters, this automatically switches the *Enabled* parameter back to *true*.

Note: The *Status* section in the constraint tooltip also shows that the current constraint is disabled.

To edit a constraint's *Enabled* parameter:

1. In the *Constraint Browser* select the constraint whose *Enabled* parameter you want to edit.
2. Click in the *Enabled* parameter value field in the *Constraint Parameter Editor* pane.
3. In the value field, select either *true* or *false* from the pull-down.

Note: If you have chosen to disable a constraint, the constraint appears as struck out text in the *Constraint Browser*.

See also [Editing the Status Parameter \(Layout Only\)](#).

Editing the Status Parameter (Layout Only)

You can set a constraint's *Status* parameter to be either *passed*, *failed*, or *not checked*.

An *passed* parameter means that the constraint has been met, while an *failed* parameter means that the constraint has not been met (not satisfied). These can either be set by an application, for example a placer or router, or can be user-specified.

- When a constraint is set to *passed*, the constraint is shown in green text in the *Constraint Manager*.
- When a constraint is set to *failed*, the constraint is shown in red text in the *Constraint Manager*.
- When a constraint has neither been set as *passed* or *failed* (that is, it's status is set as *not checked*), it is shown in the default text color, black.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Note: The above will be the case unless the constraint has been set to disabled (see [Editing the Enabled Parameter](#)), is read-only, has been overridden, or is *out-of-context*.

To edit a constraint's *Status* parameter:

1. In the *Constraint Browser* select the constraint whose *Status* parameter you want to edit.
2. Click in the *Status* parameter value field in the *Constraint Parameter Editor* pane.
3. In the value field, select either *passed*, *failed*, or *not checked* from the pull-down.

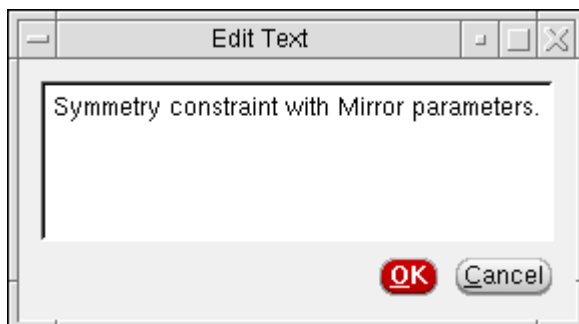
Depending on the status selection made, the constraint text gets updated in *Constraint Browser*.

Editing the Note Parameter

Each constraint in the *Constraint Manager* can have a text note associated with it. Where a note already exists, a note icon is displayed in the *Note* column of the constraint in the *Constraint Manager*.

To add or edit a constraint's *Note* parameter:

1. In the *Constraint Browser*, select the constraint whose *Note* parameter you want to edit, or add.
2. Click in the *Note* parameter value field in the *Constraint Parameter Editor* pane.
3. In the *Note* value field, either:
 - a. Add a note or edit an existing note.
or
 - b. Click the ... button to open the Edit Text form.



Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

From here, you can enter the text note manually or *Load* an existing text file. Selecting the *Load* option invokes a browser form where you can locate the text file to be added. Once you have added/edited the text note, click the *OK* button.

The new or updated note displays, partially, in the *Note* field.

If you have added a new note, a note icon appears in the *Note* column of the constraint in the *Constraint Manager*.

Constraint Parameter Editor Status Visualization

The *Constraint Parameter Editor* provides a variety of visual mechanisms to display the current status of a parameter and its current value.

Default Parameter Values

- Constraint parameters whose current values are the same as the default value are displayed in *italic blue* text. This provides an indication of which parameters have inherited their values and those parameters that have been manually altered.

Note: Those values that have been modified manually are displayed in standard, black text.

- If a parameter has been changed from its default value, a *Revert to Default* icon will be displayed as a short-cut to restore the default parameter value.

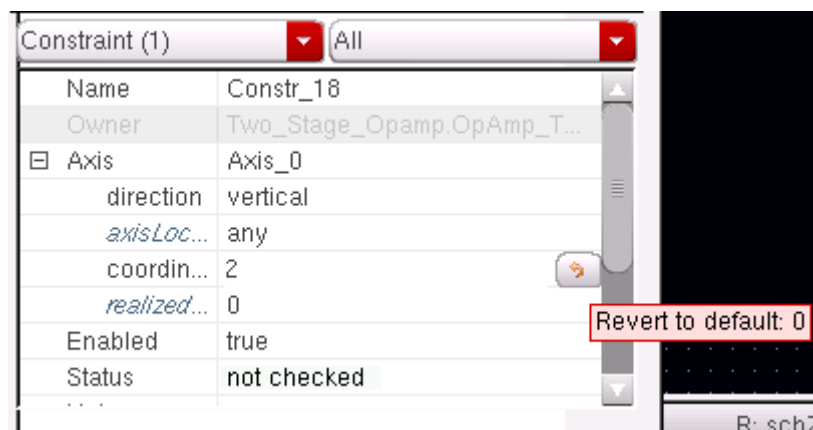
Reverting to Default Values

For certain parameters you can quickly revert back to the default parameter value.

1. Click in the value field of the parameter you want to restore the default value for.
2. If the current value is different from the default value for the parameter a red *Revert to default* icon is displayed to the right of the parameter value field.

The tooltip for the icon could be, for example, *Revert to default: 0*.

3. Click the *Revert to default* icon to automatically change the value of the current parameter back to the default value for that parameter.



Read Only Values

Parameter values that are set as read-only are displayed in gray font and cannot be edited.

Differences in Parameter Values Across Selected Objects

If there are differences in parameter values across selected constraint members, the parameter name is highlighted in red font and the parameter values summarized by the string (*various*).

Note: If there are no differences in parameter values across selected objects with the same parameter, a single value is displayed.

Transferring Constraints Between Schematics and Layout

In the forward dialog (schematic to layout transfer), constraints in the schematic can be transferred or updated in the layout on demand by using the *Update Layout Constraints* command displayed in the Constraint Manager menu in Layout XL. The constraints in the schematic hierarchy are transferred to the layout only if the layout is flat, that is, the constraint transfer stops at the layout stop points as per the physConfig. In addition, the status of the constraints in layout gets updated dynamically any time when the status of the corresponding constraints change in the schematic. The transfer and update of constraints take into account the elaboration of the components and nets in the layout (for example, m-factored or inherited instances and hierarchy expansion). On constraint transfer, the names of layout constraints bound to schematic constraints are updated to match the schematic constraints' names. Therefore, when a schematic constraint is transferred to the layout, the following checks are done:

- If the constraint name is not already in use for another constraint on the layout side, the constraint name from the schematic gets transferred as well.
- If the constraint name is already in use for another constraint on the layout side, then there is a conflict. To resolve this, the conflicting layout constraint gets auto-renamed to allow the transfer.

In the backward dialog (layout to schematic transfer), constraints in the layout can be transferred or updated in the schematic on demand. The status of the constraints in schematic are dynamically updated any time when the status of the corresponding constraints change in the layout.

Important

If Virtuoso Design Intent is implemented, to ensure that a constraint view is not created when transferring constraints from Layout XL to Schematics XL using the *Update Schematics/Layout Constraints* button, all design intents implemented using the constraint manager must have their constraints included in the design intent template (physical constraint). For more information, see the [Virtuoso Design Intent User Guide](#).

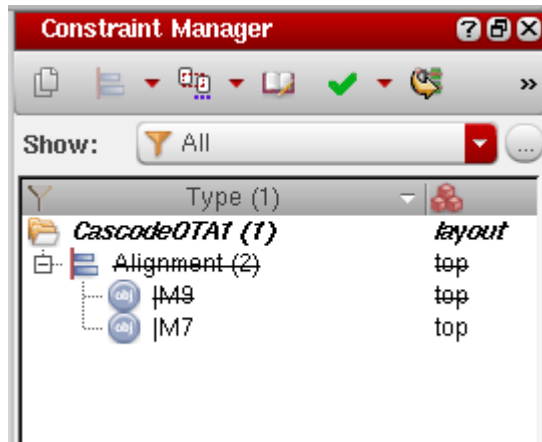
The constraint view list fields in the [Cadence Hierarchy Editor](#) (HED) and the Configure Physical Hierarchy (CPH) utility allow you to configure constraints for a design hierarchy for the purpose of simulation and physical implementation, respectively.

To generate layout representations of the schematic design components, select the *Connectivity – Generate – Selected From Source* option.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

If any **member** is transferred from schematic to layout, its associated constraint also get transferred. However, if any member of the constraint is missing, the constraint name appears as struck out in the Constraint Manager assistant in Layout XL, as shown below:



In addition, if any **instance** is transferred from schematic to layout, its related constraint and nets also get transferred. For more details, see [Defining a Constraint List at the Global Level](#) and [Constraints in Layout](#).

For further information, see the following:

- The transfer of constraints from logical to physical see [Constraint Transfer](#) and [Configuring the Physical Hierarchy](#) in the Virtuoso Layout Suite documentation.
- Constraint updates see [Updating Design Data](#) in the Virtuoso Layout Suite documentation.
- Physical design constraints and their uses in Layout XL (and higher tiers), see [Constraint Manager in Layout XL](#) in the Virtuoso Layout Suite documentation.
- Working with constraints and design intents, see [Virtuoso Design Intent User Guide](#).
- Reporting constraint differences between two designs (schematic source design and layout target design), see the [ciPrintReport](#) SKILL function in the [Virtuoso Unified Custom Constraints SKILL Reference](#).

For general information on constraint management in Layout XL (and higher tiers) see [Constraint Manager in Layout XL](#).

Using Schematic Constraints in Layout

Enabled constraints can be transferred from schematics to layout in a *logical to physical transfer*.

Constraints are transformed during the transfer by mapping the schematic members to the corresponding objects in the layout, taking account of m-factor, fingers, and other implementation choices.

Note: A transferred constraint is considered to be out-of-context if any of its members are not present in the layout. If all constraint members are out-of-context, that is, none of the constraint members exist in the layout, the constraint is not shown in the layout *Constraint Manager*.

Once constraints have been transferred to layout, the layout designer can create additional constraints on the design, enable or disable constraints, or add notes to them. Doing all of this may help resolve or avoid issues where some constraints cannot be met because of conflicting requirements, particularly when using automated software applications.

For more information see Constraint Transfer in the Virtuoso Layout Suite documentation.

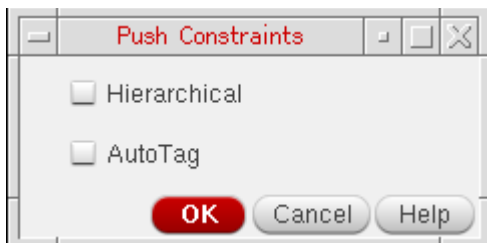
Pushing and Pulling Constraints in a Schematic or Layout Hierarchy

You can push and pull constraints and their parameter members on a schematic or layout hierarchy by using the *Constraint Propagation* options. These are available on the *Constraint Manager* toolbar in Schematics XL and in VLS XL (and higher tiers) (only).

Note: This functionality relies on connectivity to propagate constraints through nets in the hierarchy, therefore this functionality is available only in VLS XL (and higher tiers).

You can specify whether to push up or pull down these constraints in the hierarchy using the *above* and *below* values of the *hierarchicalScope* parameter. The setting, *local* is mandatory and set by default. To propagate custom constraints and their member parameters, this parameter needs to be specified as an `enumSet` type.

Note: You can also tag constraints for propagation automatically, without having to explicitly set the scope parameter by checking the *AutoTag* option in the propagation dialog box. This is displayed after choosing *Push Constraints* or *Pull Constraints* from the *Constraint Propagation* drop-down list.



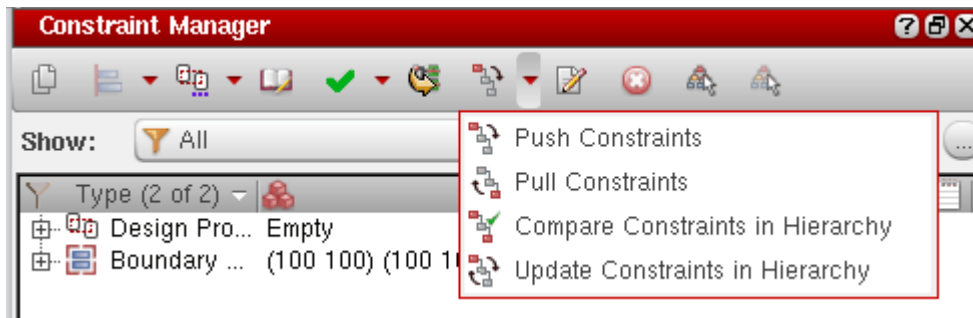
The constraint propagation works on selected instances. However, if nothing is selected, constraints propagates to or from all instances at the next level of the hierarchy.

Note: Ensure that the net to be pushed or pulled is connected to the `instTerm` of the selected instance.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

After the appropriate constraints have been specified, choosing *Push Constraints* or *Pull Constraints* from the *Constraint Propagation* drop-down list propagates these constraints as required.



Note: When a constraint is pushed or pulled, the constraint name is preserved if possible. If a constraint with that name already exists, the next available name is generated by keeping the source constraint name as the basename and adding a counter to it. For example, if there is a constraint named `myConstraint` that needs to be pushed and another constraint with the same name already exists, then a counter can be added to the pushed constraint's name like `myConstraint_1`, `myConstraint_2`, `myConstraint_3`, and so on depending on availability. If you need to extract the basename from the names of such constraints, you can use the `ciConBaseName` SKILL function.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

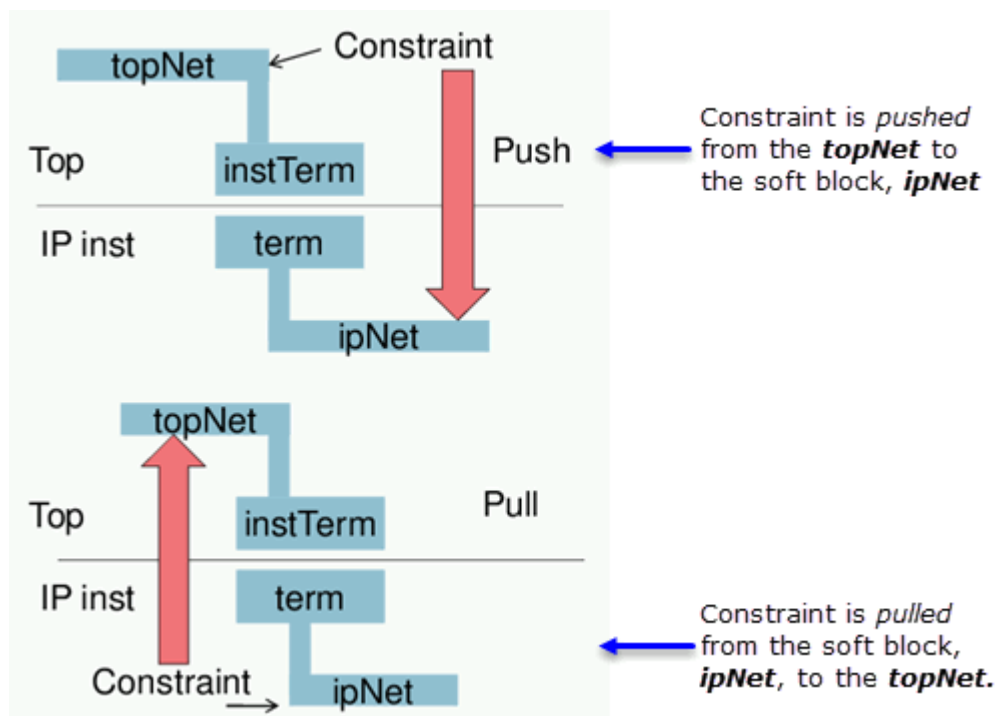
The *Constraint Propagation* drop-down list has the following options:
Following is an example use of pushing and pulling constraints in a soft block:

Option	Description
<i>Push Constraints</i>	Specified constraints are copied (pushed) to softblocks and schematic/layout cells lower in the design. On completion, the copied constraints' members, such as nets and terminal instances, correspond to the original constraint members because they are physically connected. For Layout only, see also ciPushConstraints and ciPushConstraint. Push constraints can also be propagated recursively either to the bottom or the top of the hierarchy from the Constraint Manager. To do this, you need to click  from the Constraint Manager assistant toolbar. You can also tag constraints for propagation automatically without having to explicitly set the scope parameter by checking the <i>AutoTag</i> option in the propagation dialog box.
<i>Pull Constraints</i>	Specified constraints are copied (pulled) from blocks lower in a design up to the top design. On completion, the copied constraints' members, such as nets and terminal instances, correspond to the original constraint members because they are physically connected. For Layout only, see also ciPullConstraints and ciPullConstraint. Similarly, Pull constraints can be propagated recursively either to the bottom or the top of the hierarchy from the Constraint Manager. To do this, you need to click  from the Constraint Manager assistant toolbar. You can also tag constraints for propagation automatically without having to explicitly set the scope parameter by checking the <i>AutoTag</i> option in the propagation dialog box.
<i>Compare Constraints in Hierarchy</i>	Compares the pushed and pulled constraints with the source constraint. For Layout only, see also ciHierCompareConstraint and ciHierCompareConstraints.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Option	Description
<i>Update Constraints in Hierarchy</i>	Updates the pushed and pulled constraint as per the changes made in the source constraint and vice versa. Note: For Layout only, there is an optional <i>update</i> argument in both the <code>ciPushConstraint(s)</code> and <code>ciPullConstraint(s)</code> APIs.



Note: Here, IP inst is an instance, while ipNet refers to a lower level net.

Constraints vs. Rules

Process Constraints

Process constraints are technology specific rules typically stored in a technology file, for example constraints concerning the minimum or maximum allowable width for a given layer.

The design rule driven capability (*DRD) in the Virtuoso Layout Suite provides the necessary interactive assistance in this regard and the automatic applications (placer, router, and optimizer) adhere to these rules.

Note: *When DRD markers exist in a design, they should be read, processed, and then have their Constraint Status updated to ensure that their display setting is current.

Additionally, in some cases, a foundry may provide multiple rule values.

Design Constraints

Design constraints are user-defined rules that are set to meet specific design goals such as performance and reliability, typically specified by the circuit designer.

Examples of design constraints include:

- Net based width/length or IR drop constraints for timing/power.
- Matching differential pairs for analog circuit performance.
- Spacing/shielding constraints that are driven by signal integrity requirements.

Layout Specific Rules

Layout specific rules are rules that enable a more compact layout and help to build the layout incrementally. For example, grouping or fixing some (placed and routed) components in the design during ECO. These may or may not be driven by Design Constraints.

Note: For more specific information on physical design constraints and their uses in Layout XL, see Constraint Manager in Layout XL.

Process Rules for Objects

By default, the process rules for all objects are derived from the foundry constraint group in the technology database. These rules can however be overridden for specific objects or for a given design.

Process rule overrides on any object apply to other objects following a hierarchical precedence. For example, when applied to nets they will affect all routes and shapes that belong to that net, and when applied to routes they will affect all the shapes on that route. In Virtuoso, applications follow the precedence of: *wires* (paths/pathSegs/vias/guides) -> *routes* -> *nets* -> *net classes* -> *wire edit constraint group* -> *design*, when looking up constraints on objects (defaulting to foundry). The lookup is further guided by the ordering of constraints within any one `constraintGroup` and the precedence ordering of the constraint groups themselves.

Note: The `ciAddProcessRules` SKILL function only applies the *hard* lookup find, that is only to those constraints that are within any single `constraintGroup`.



You are encouraged to use CST Access SKILL Functions as they will allow for access to process rules directly using SKILL.

Example of Use

To determine `Metall1 minWidth` on a `pathSeg`, the route that contains the `pathSeg` is checked first. If a value is found, the lookup stops, otherwise a value for `Metall1 minWidth` is searched for on the net that contains the `pathSeg`. Again, if found, the search stops. Otherwise, it continues up to larger container objects, with the last place searched being the technology database foundry constraint group. Also, while looking for a constraint on the object it is possible that more than one `Metall1 minWidth` constraint is found. The process rules applied to an object have an order, and during lookup they are traversed in that order. If a group is encountered in the object's rule list, it is traversed completely before continuing to the next element as shown below:

```
C = constraint
G = group
Indent = group membership
C1 Metall1 minWidth
C2 Metall1 minWidth
G1 group
    C3 Metall1 minWidth
    C4 Metal2 minWidth
C5 Metal2 minWidth
```

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Based on the above, the value for `Metal1 minwidth` is taken from constraint C1 while the value from `Metal2 minWidth` is taken from constraint C4.

See also [The Process Rule Editor](#).

Using Constraint Notes



The constraint notes functionality is only available in Virtuoso Schematic Editor XL.

Constraint notes can be added to a design canvas to assist design reviews, and enhance communication between front- and back-end engineers, as well as acting as a source of future reference.

In Schematic Editor XL, while working on a writable schematic design, you can generate system-defined and user-defined constraint notes for existing and new constraints, and user-defined constraint notes for a template. This can be done either automatically or manually.

The system-defined constraint notes are created for a restricted list of constraint types that comprise *Symmetry*, *Matched Parameters*, *Distance*, *Orientation*, *Modgen*, *Cluster*, *Guard Ring*, and *Cell Boundary*. For more information, refer to the [System-Defined Constraint Notes](#) section.

On the contrary, the user-defined constraint notes can be created for all system-defined and user-defined constraint types. For information on how to create your own, customized constraint notes, refer to the [User-Defined Constraint Notes for Constraints](#) section.

The capability to generate user-defined constraint notes for a template is useful because a template reflects the design intent that should be recognizable to the designers. Optionally, you can toggle between the constraint note for a template and the constraint notes for the underlying constraints. For more information, refer to the [User-Defined Constraint Notes for Templates](#) section.

System-Defined Constraint Notes

The following sections discuss how to use system-defined constraint notes in automatic mode and manage them using the menu options.

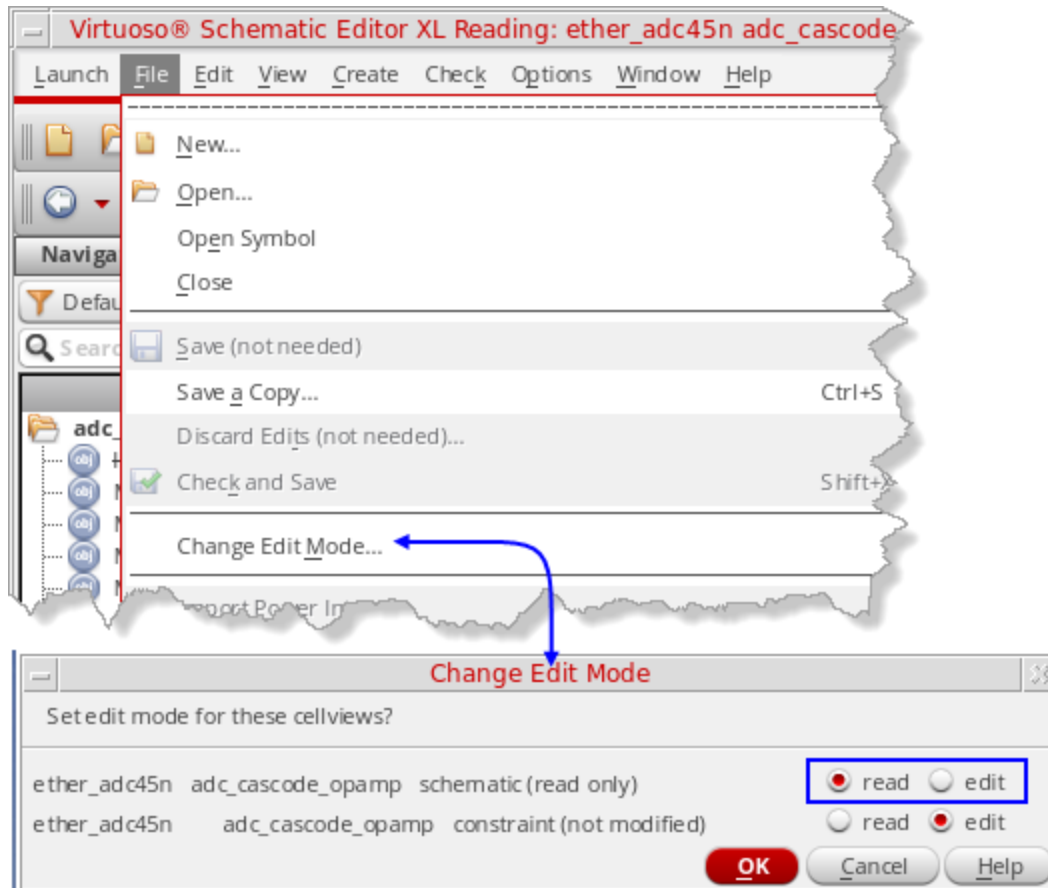
Setting Up Automatic Generation of Constraint Notes

Using the automatic generation of constraint notes, you can avoid manual management of the constraint notes. When you use this mode, the constraint notes can be automatically applied to the existing constraints.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

To enable automatic generation and update of constraint notes, it is important that the schematic view is opened in edit mode. You can check this from the *File – Change Edit Mode* menu, clicking which displays the *Change Edit Mode* form as following:



Then perform the following steps:

1. Select *Options – Editor* in VSE XL.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

The Editor Options form is displayed.

The screenshot shows the 'Editor Options' dialog box with the 'Commands' tab selected. The 'Command Controls' section includes settings for instance browser type, schematic views, pin direction rule, wire snapping, solder dots, wire overlap, pin sorting, symbol pins, repeat commands, cross selection, gravity, marker size, view name list, and bus terminals. The 'Constraint Notes Controls' section, highlighted with a blue box, includes 'Auto Constraint Notes' (set to 'disable'), 'Sync on Design Open' (set to 'disable'), and 'Sync on Design Check' (set to 'disable'). The 'OK' button is highlighted in red.

2. In the *Constraint Notes Controls* group box, select the *enable* radio button to specify whether the *Auto Constraint Notes* control should be used. This helps in automatic generation of constraint notes.



Tip

Once constraint notes have been enabled on your design, you can toggle on or off their display in the canvas by selecting *View – Show/Hide Notes*.

3. When you enable automatic generation of constraint notes, the inactive radio buttons adjacent to *Sync on Design Open* become available for selection. By default, *disable* is selected. If you want to synchronize the constraint notes between schematic and constraint views when you open the design or change the mode of the constraint view, select the *enable* radio button. Alternatively, set the `constraintNotesSyncOnDesignOpen` environment variable to `t`.
4. The *Sync on Design Check* control is also disabled by default. Select the *enable* radio button adjacent to this control if you want the constraint notes to be synchronized on *Update Connectivity*. Alternatively, set the `constraintNotesSyncOnDesignCheck` environment variable to `t`.

Enabling Constraint Note Creation Using `autoConstraintNotes`

Alternatively, you can enable automatic management of constraint notes by using the `autoConstraintNotes` environment variable as following:

```
constraint autoConstraintNotes boolean t
```

Choosing the *enable* option in the Editor Options form automatically sets this environment variable to `t` for new constraints, and enables the creation of notes for the existing constraints. However, if the constraint notes environment variable is set through the CIW, notes will be created only for the new constraints, not the existing ones. If you still want to use the CIW to enable constraint notes, it is recommended that you do this through `.cdsinit`, or before launching the tool, so that all constraints notes are created when the tool is launched.

Managing Constraint Notes Manually

If you have not set up constraint notes to be created automatically, you can choose to manage them either using the right-mouse-button contextual-menu in the design canvas, or using the Create – Note menu options.

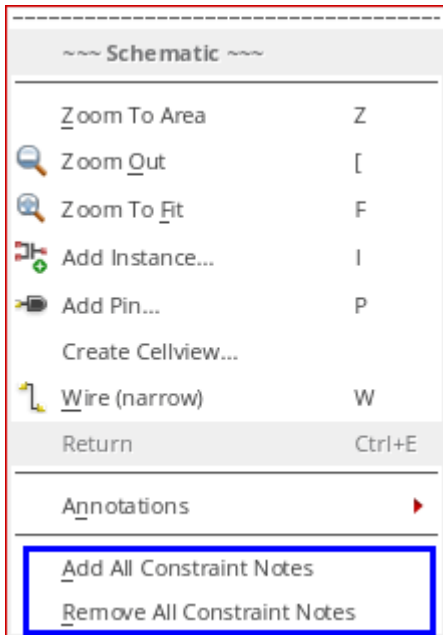
Using the Context Menu to Create, Update, and Remove Constraint Notes

The contextual-menu options displayed will depend upon objects under constraint being currently selected in the design canvas, constraints being selected in the *Constraint Manager*, or whether no such objects are currently selected. For information about the context menus available in Constraint Manager, refer to Constraint Browser Context-Menus.

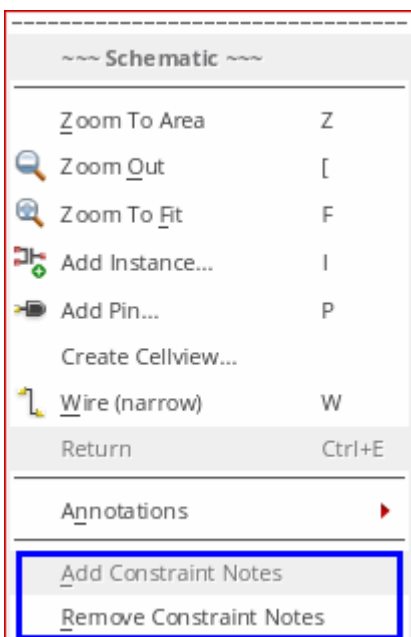
Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

- If **no objects (under constraint)** are currently selected in the design canvas, clicking the right-mouse-button (RMB) in the design canvas displays the following contextual-menu:



- If **one or more constraints** are currently selected in the Constraint Manager, clicking the RMB in the design canvas will display the following contextual menu:



Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

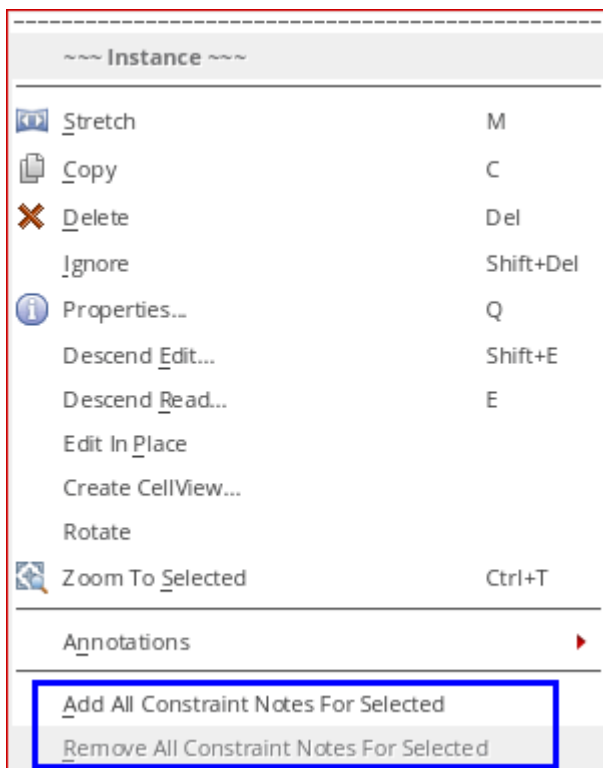
Additionally:

- ❑ If all of the currently selected constraints, in the *Constraint Manager*, already have constraint notes, then *Add Constraint Notes* is disabled.
- ❑ If none of the selected constraints have notes, *Remove Constraint Notes* is disabled.
- ❑ If some of the constraints of the selected constraints have constraint notes, both the *Add Constraint Notes* and *Remove Constraint Notes* options are enabled.

Important

The enabling of the *Add Constraint Notes* option is also dependent upon the selected constraints (or additional constraints that have been added by customized scripts) being able to be annotated. For information on those constraints that can be annotated, see [Using Constraint Notes](#).

- If **one or more constrained objects are currently selected (or a mix of constrained objects and constraints) in the design canvas**, clicking the RMB will display the following contextual-menu.



Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Note: For comparable information on the enabled and disabled states of the *Add All Constraint Notes For Selected* and *Remove All Constraint Notes For Selected* options, see [here](#).

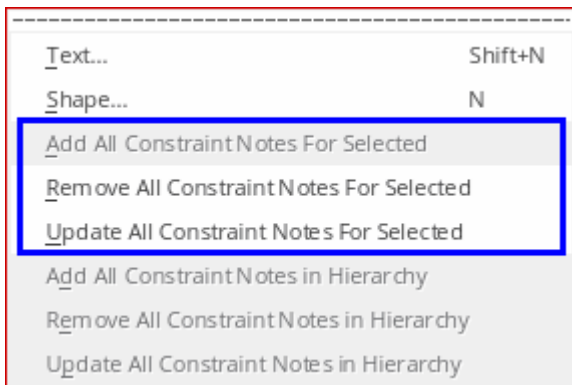
Using the Create Menu to Create, Remove, or Update Constraint Notes

You can also use the following *Note* sub-menu options in the *Create* menu for manually controlling constraint notes.

Note: For information on the enabled and disabled states of the options below, see [here](#). For example, for the *Remove...* and *Update...* options to be enabled, a constraint note has to be present for one or more of the selected objects/constraints.

The *Note* sub-menu options below will be displayed **if one or more constrained objects are currently selected in the design canvas**.

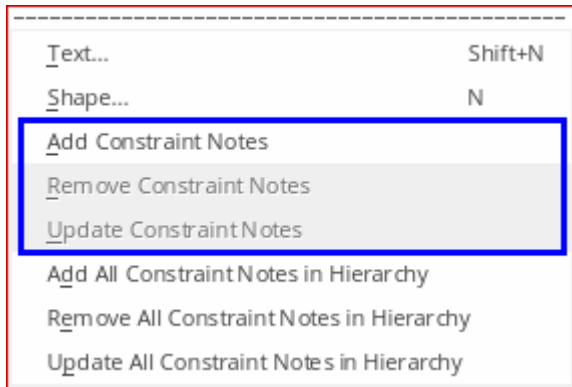
- *Create – Note – Add All Constraint Notes For Selected*
- *Create – Note – Remove All Constraint Notes For Selected*
- *Create – Note – Update All Constraint Notes For Selected*



Virtuoso Unified Custom Constraints User Guide

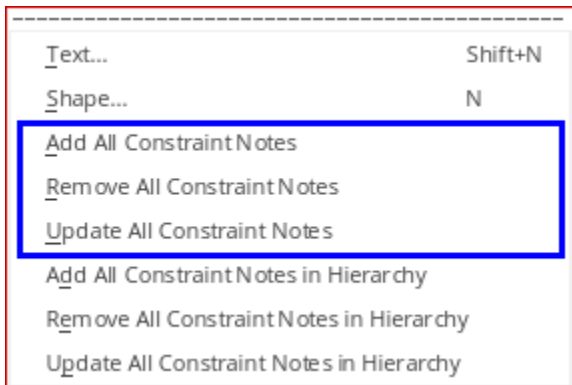
The Constraint Manager Assistant

If one or more constraints are selected in the Constraint Manager, the following *Note* sub-menu options are displayed:



If there are no current object selections, the *Note* sub-menu options displayed will be:

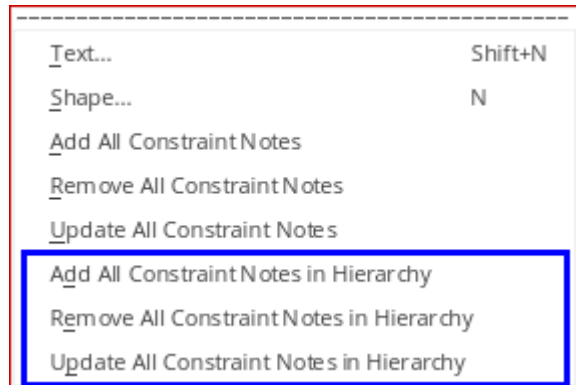
- *Create – Note – Add All Constraint Notes*
- *Create – Note – Remove All Constraint Notes*
- *Create – Note – Update All Constraint Notes*



If you want to add, remove, or updated all objects in the hierarchy, you need to select one of the following *Note* sub-menu options:

- *Create – Note – Add All Constraint Notes in Hierarchy*
- *Create – Note – Remove All Constraint Notes in Hierarchy*

■ *Create – Note – Update All Constraint Notes in Hierarchy*



Examples of Creating Constraint Notes for Different Constraint Types

Important

The following sections detail examples of how to create constraint notes for a range of constraint types. It is assumed that the *Auto Constraint Notes* option has been enabled in the Editor Options form, although you can also use the manual approach to achieve the same results.

This section details how to create constraint notes for the following constraint types:

- Symmetry
- Matched Parameters
- Distance
- Orientation
- Modgen
- Cluster
- Guard Ring
- Cell Boundary

Overview Example of Schematic Cellview with Constraint Notes Applied

The following schematic cellview allows you to visualize numerous constraints and their already applied constraint notes.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

The schematic cellview contains notes for the following constraints: 1 *Cell Boundary*, 3 *Modgen* (each with a *Guard Ring* applied), 2 *Clusters* (one of which has a *Guard Ring*), 1 *Matched Parameter*, 2 *Orientation*, and 9 *Symmetry*.

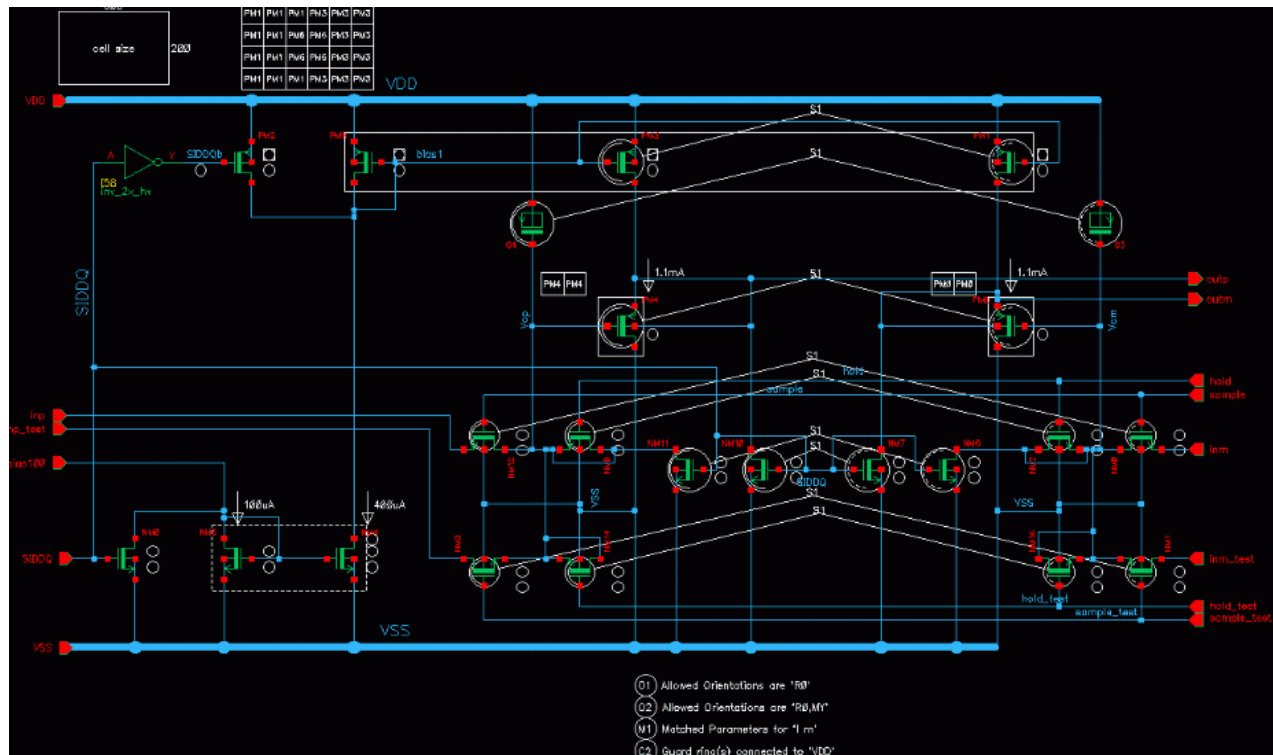
Note: For all of the above constraint notes, the first character of any *reference number* will be the first character of the constraint type name. When that first character is in upper-case, it indicates that all members are located in the current cellview. When the first character is in lower-case, this indicates that at least one constraint member is located in a sub-cell. The legends for constraints that have members in multiple instances, are created at the bottom left of the current cellview, while those legends for constraints that only have members in the current cellview will be created in the bottom right corner.



Tip

You can select *View – Hide Instance Labels*, in VSE XL, to aid visualization of constraint notes.

The following figure shows a schematic cellview with many constraint notes applied:



Details on how to apply all of the constraints notes shown in the above schematic cellview will now be covered in the remainder of this section.

Note: For information on how to create your own, customized, constraint notes, see [User-Defined Constraint Notes for Constraints](#).

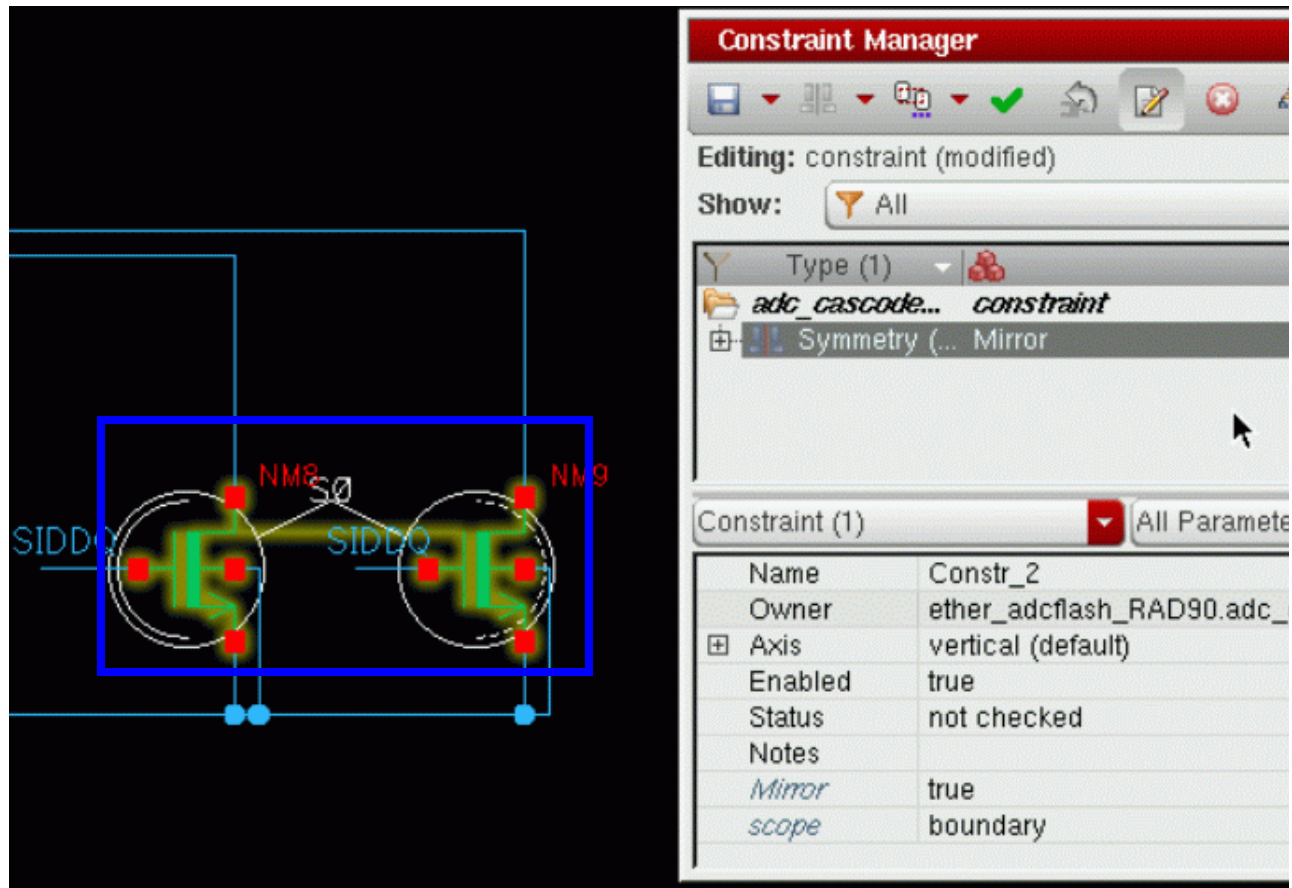
Creating Constraint Notes for a Symmetry Constraint

Example steps for creating a [Symmetry](#) constraint note are as follows:

1. In VSE XL, open a schematic that comprises of primitive components, for example, transistors, diodes, and passive devices.
2. Ensure that *Auto Constraint Notes* is set to *enable* (*Options – Editor*).
3. If not already displayed, open the Constraint Manager assistant (*Window – Assistants – Constraint Manager*).
4. In the design canvas, select a pair of devices and create a *Symmetry* constraint (see [Creating Constraints in the Constraint Manager](#)).

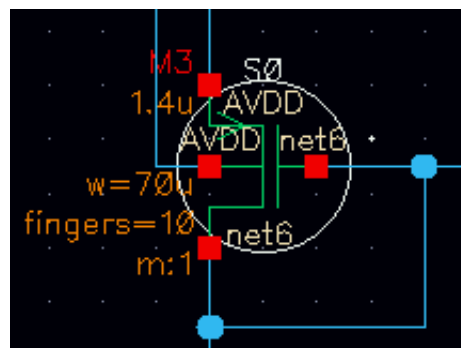
As part of this *Symmetry* constraint creation, a related graphical note will also be created.

Figure 1-20 Constraint Note Automatically Added to New Symmetry Constraint



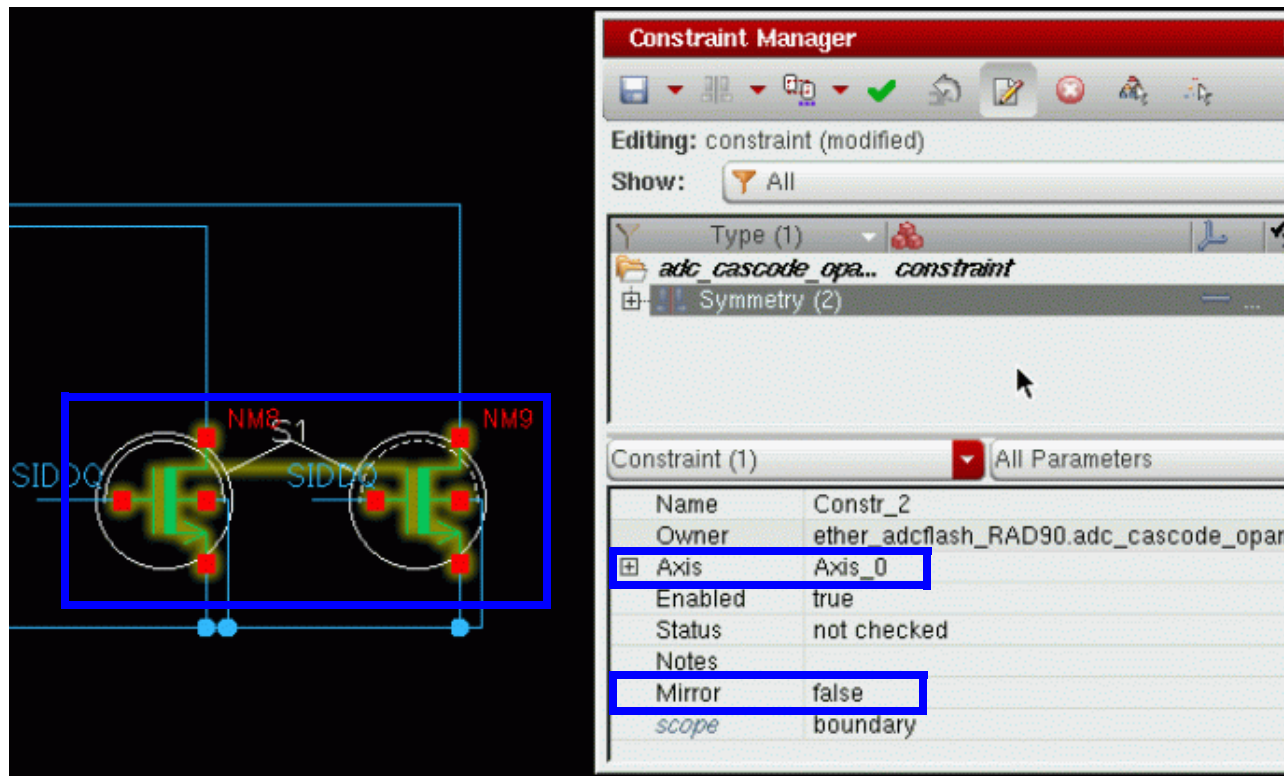
The constraint note will also update if the constraint parameters are edited. For example, in a *Symmetry* constraint, if the *Axis* direction or *Mirror* value is changed.

Figure 1-21 Constraint Note for a Self-Symmetric Instance



A reference number ($S0$) is also shown for self-symmetrical instance constraints as it can provide information as to whether two self-symmetric instances share the same axis.

Figure 1-22 Constraint Note Updated as Parameters Have Been Edited



Understanding Symmetry Constraint Notes

For symmetry constraint notes, the variants of the pictogram are defined as follows:

- A reference number ($S0$, $S1$, ...) is attached to two solid circles, surrounding each member.
Note: For self-symmetry, the unique member is surrounded by a single circle.
- When the semi-circle inside the circle is solid, the member is considered to be the reference member. The semi-circle is dashed for the second member.
- When both tips of the semi-circles are aligned vertically, the symmetry axis is vertical. However, when both tips are aligned horizontally, the symmetry axis is horizontal.
- When both semi-circles are mirror orientated, the symmetry is mirrored, otherwise the symmetry is simple.

- When the reference number is the same for two symmetry constraints, the constraints share the same symmetry axis, otherwise they have different symmetry axes.

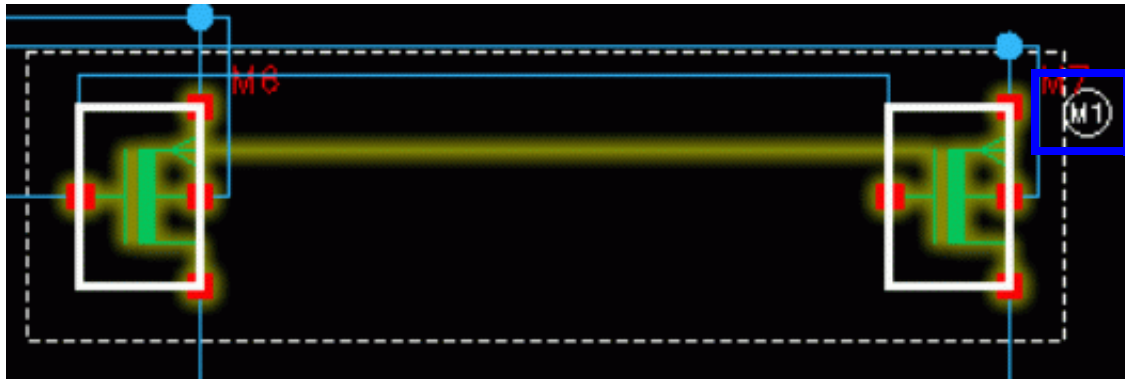
Creating Constraint Notes for a Matched Parameters Constraint

Example steps for creating Matched Parameters constraint notes are as follows:

1. In VSE XL, using an appropriate schematic design, select a pair of devices and create a *Matched Parameter* constraint.

A new *Matched Parameter* constraint note will be created simultaneously.

Figure 1-23 Constraint Note for a Matched Parameter Constraint



Understanding Matched Parameter Constraint Notes

When all members are aligned in the schematic, a rectangle is drawn around all of the *Matched Parameter* constraint members. Additionally, a legend note will be added at the bottom of the schematic.

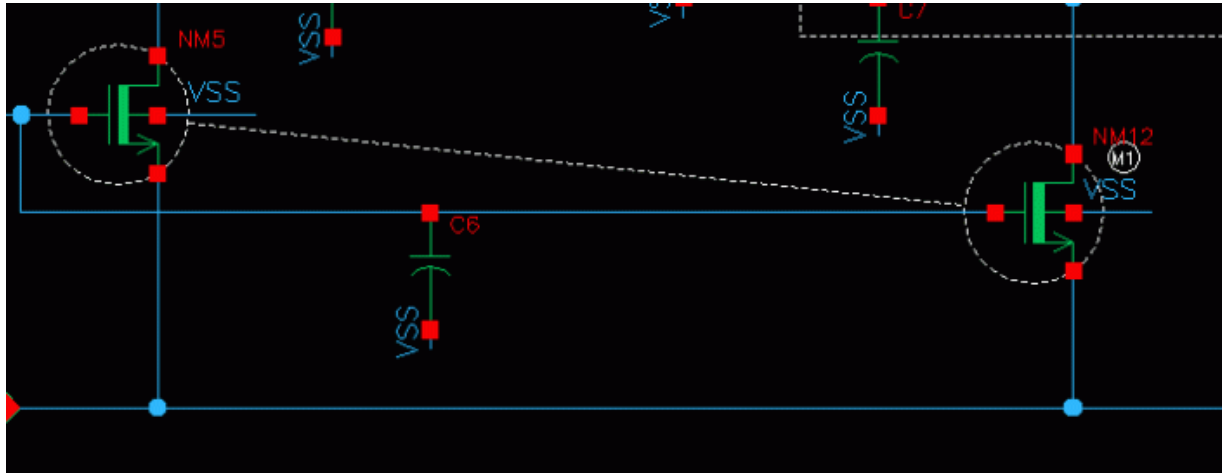
Figure 1-24 Matched Parameter Legend Note



Note: The encircled **M1** reference number, located close to the member (see [Figure 1-23](#) on page 207), makes reference to the corresponding legend at the bottom of the schematic. To avoid clashes in the schematic, with too much information located close to the members, the reference number and legend allows for a clear extension of such information.

- If the *Matched Parameter* constraint members are not aligned, the constraint pictogram note comprises of dashed circles around the members, linked with a dashed line segment and a reference number.

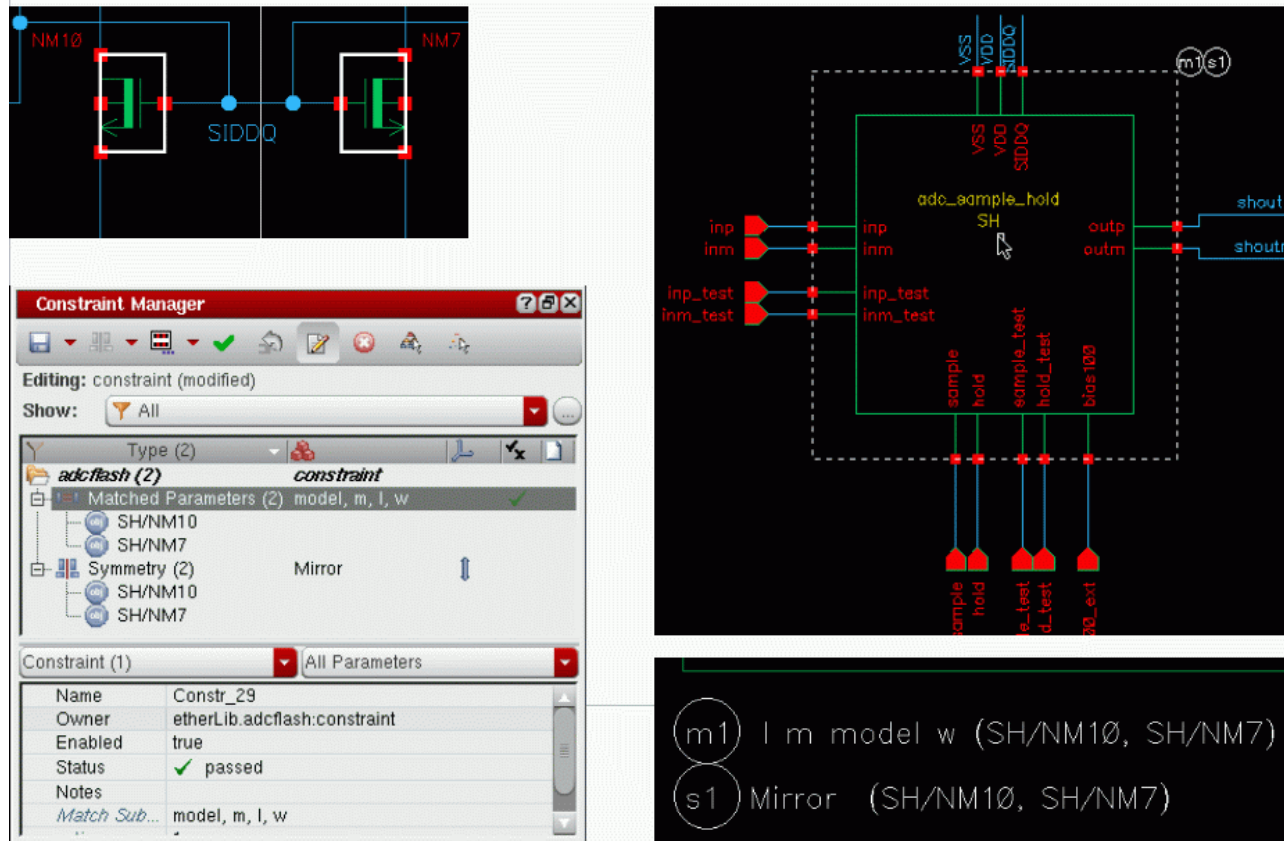
Figure 1-25 Constraint Note for Non-Aligned Matched Parameter Constraint



- When the members of a *Matched Parameter* constraint are selected in a sub-cell, the constraint note will comprise of a reference number, and the corresponding legend will be created at the bottom left of the cellview.

Note: In the example below, *Symmetry* and *Matched Parameter* constraints have been set to the same pair of NM10 and NM7 instances in the SH instance.

Figure 1-26 Constraint Members Selected in Sub-Cell



Creating Constraint Notes for a Distance Constraint

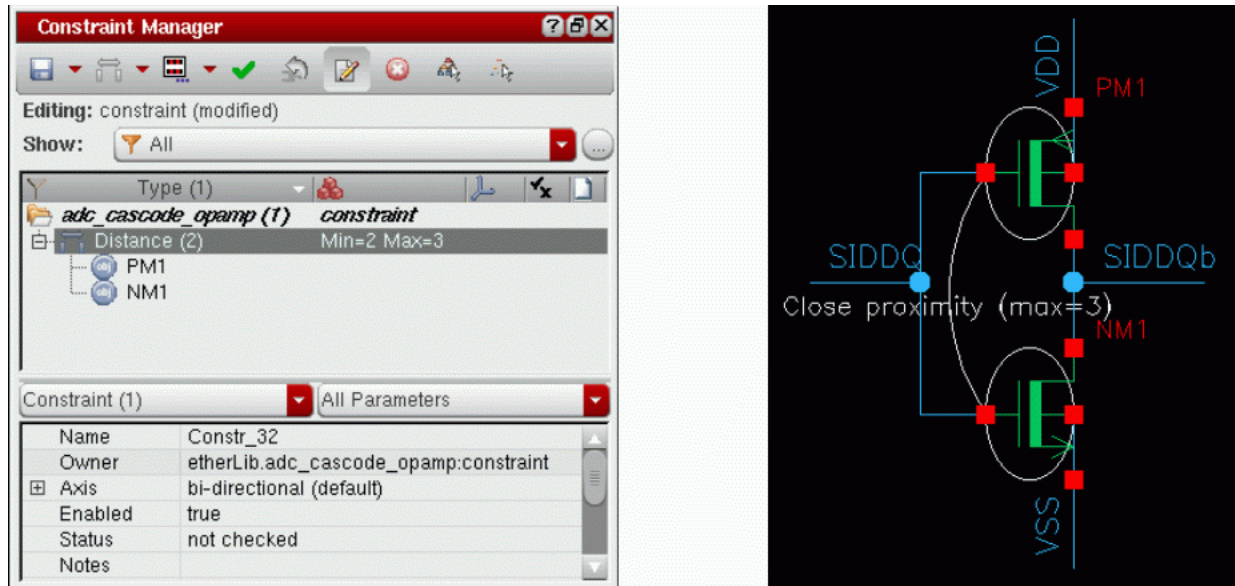
Example steps for creating a Distance constraint note are as follows:

1. In VSE XL, using an appropriate schematic design, select a pair of instances and create a *Distance* constraint.

On creation, each *Distance* constraint member will be surrounded by an ellipse, with each ellipse linked with an arc and a text note showing the maximum value.

Note: When a *Note* parameter is updated, the constraint note text will be replaced by the contents of the string value entered.

Figure 1-27 Constraint Note Created for a Distance Constraint



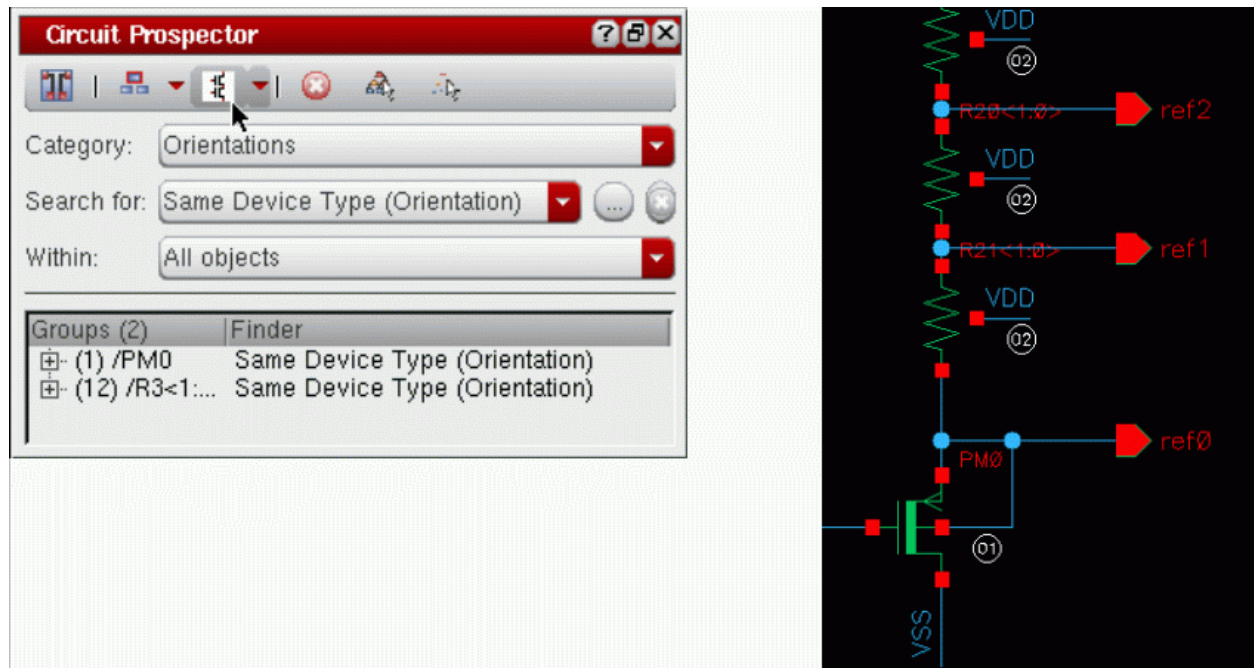
Creating Constraint Notes for an Orientation Constraint

Example steps for creating a Orientation constraint note are as follows:

1. In VSE XL, using an appropriate schematic design, create *Orientation* constraints for a complete schematic using the *Circuit Prospector* assistant (select *Window – Assistants – Circuit Prospector* if it is not already on display).
2. In the *Circuit Prospector* assistant, set the *Category* to *Orientations*, and the *Finder* to *Same Device Type (Orientation)*.
3. Click the *Generate Orientation Constraint per Device Type for the Current Cellview* option in the *Circuit Prospector* toolbar.

This will set a default *Orientation* constraint, per device type.

Figure 1-28 Orientation Constraint and Constraint Notes Created Using the Circuit Prospector Assistant



As *Orientation* constraints are common for all custom layouts, only a reference number is set close to each instance member. A legend will be created below the current cellview, informing of which orientation values are allowed for the corresponding instances.

Figure 1-29 Constraint Note Legend for Orientation Constraint

AUTHOR :	
REVISION :	DATE :
01 Allowed Orientations are 'R0,MY' 02 Allowed Orientations are 'R0,R180,MY,MX'	

Creating Constraint Notes for a Modgen Constraint

Example steps for creating a Modgen constraint note are as follows:

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

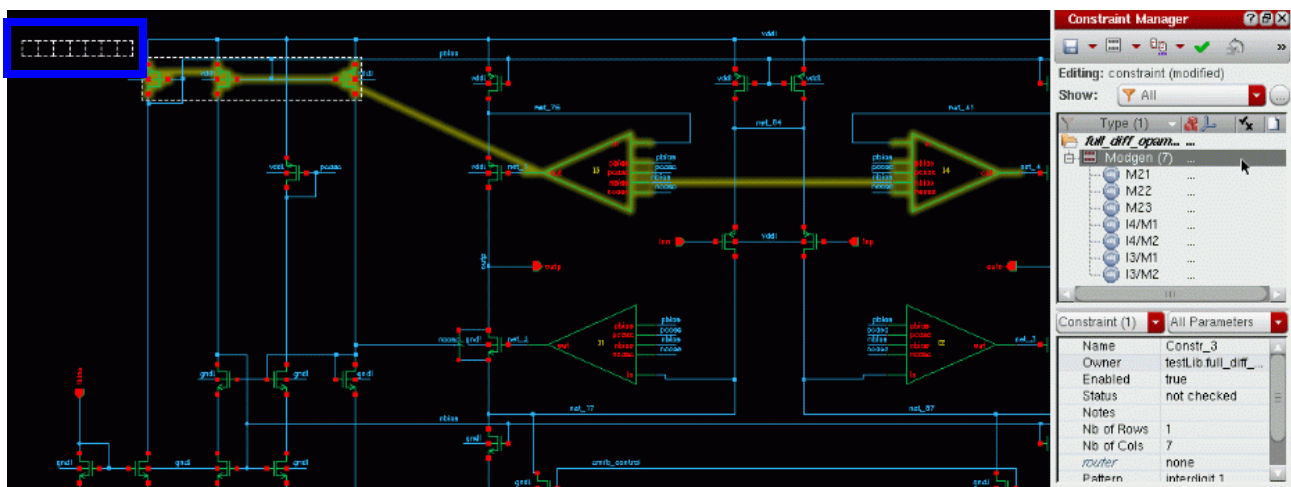
1. In VSE XL, using an appropriate schematic design, select a number of instances in the current cellview, and also some additional instances in sub-cells.

This will ensure that you obtain a packed array for devices that require good matching.

2. Create the *Modgen* constraint in the *Constraint Manager*.

A constraint note will automatically be created, showing all members and how they are interdigitated together.

Figure 1-30 Constraint Note Created for a Modgen Constraint



Note: If the *Modgen* has a guard ring, the constraint note will comprise of solid lines.

Creating Constraint Notes for a Cluster Constraint

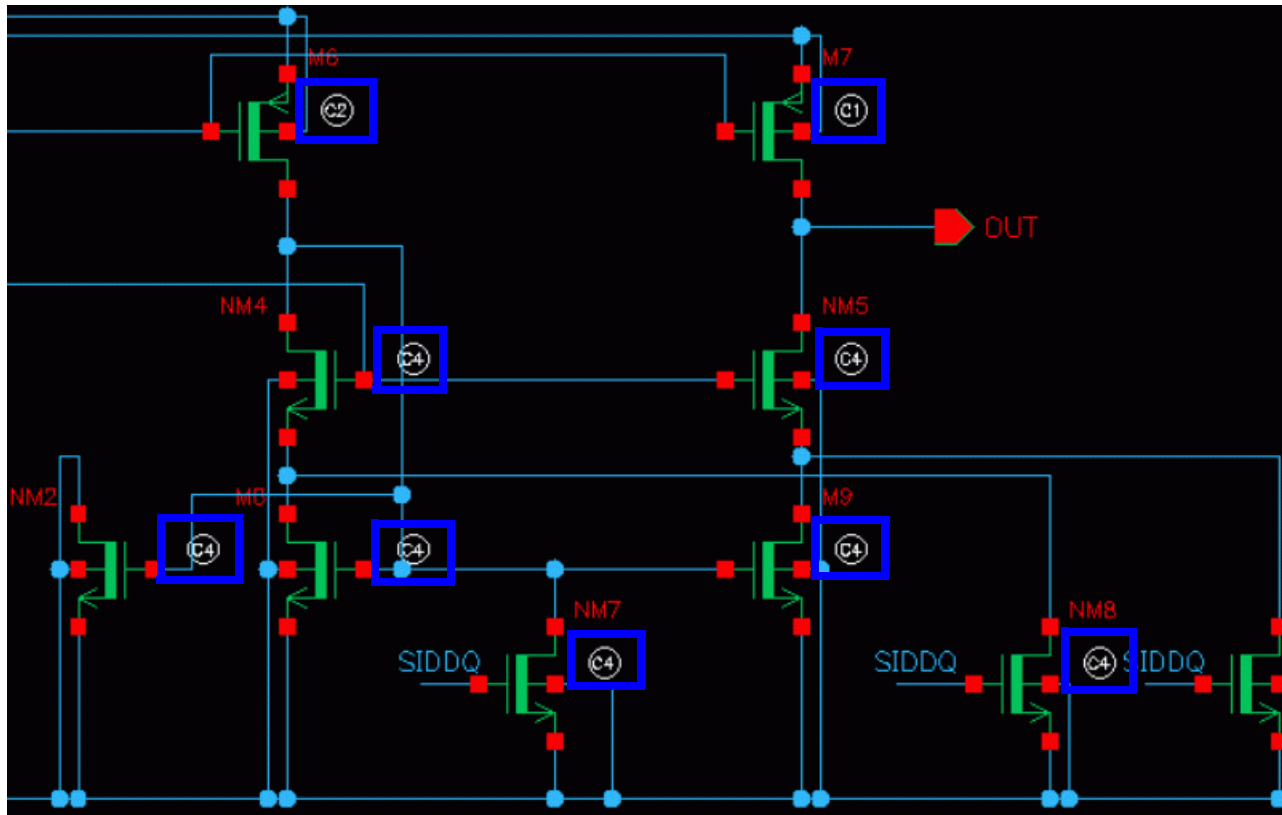
Cluster constraints can be set against device groups, with the intention of placing them in the same well. Example steps for creating a *Cluster* constraint note are as follows:

1. In VSE XL, using an appropriate schematic design, create a *Cluster* constraint using the *Circuit Prospector* assistant (select *Window – Assistants – Circuit Prospector* if it is not already on display).
2. In the *Circuit Prospector* assistant, set the *Category* to *Active Devices*, and the *Finder* to *Active Same Cell and Common Bulk*.
3. Click the *Generate Cluster Constraint for the Current Cellview* option in the *Circuit Prospector* toolbar.

The *Cluster* constraint will be created along a constraint note which comprises of a reference number.

All instances in the same cluster will have the same reference number applied.

Figure 1-31 Cluster Constraint Notes



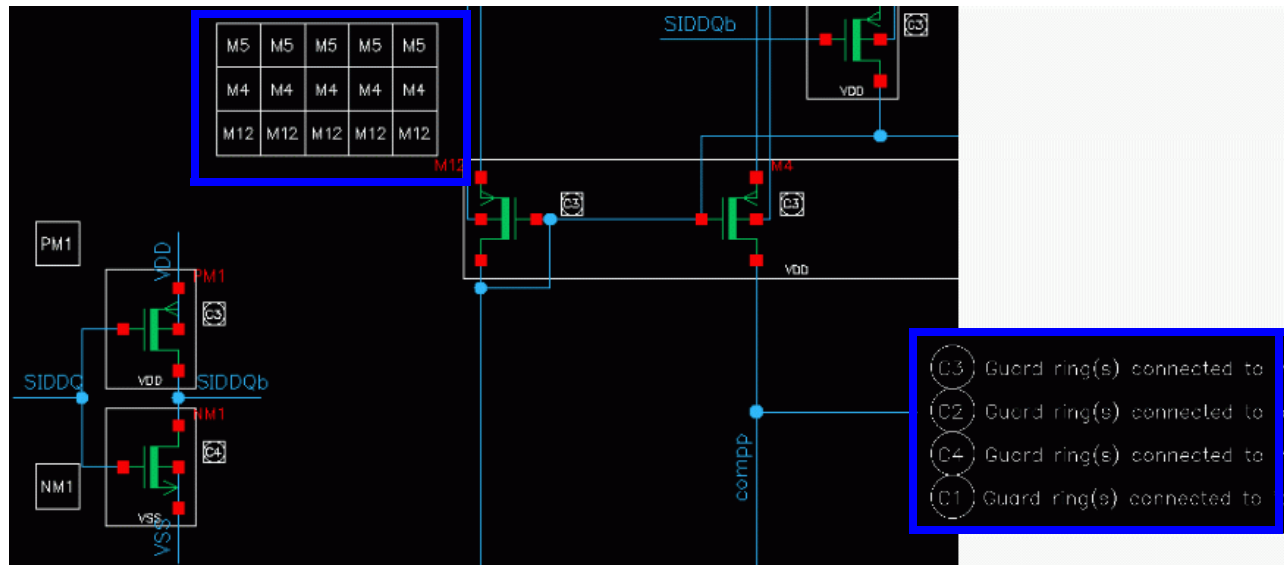
Creating Constraint Notes for a Guard Ring Constraint

Guard Ring constraints can be set for *Modgen* or *Cluster* constraints.

When a *Guard Ring* constraint is set, the array note for a *Modgen* will be made of solid lines, and the net name for the *Guard Ring* will be noted in the rectangle that surrounds the constraint members.

Alternatively, a reference number for a *Cluster* will be surrounded by a solid square, and the net name noted in the legend.

Figure 1-32 Guard Ring Constraint Note



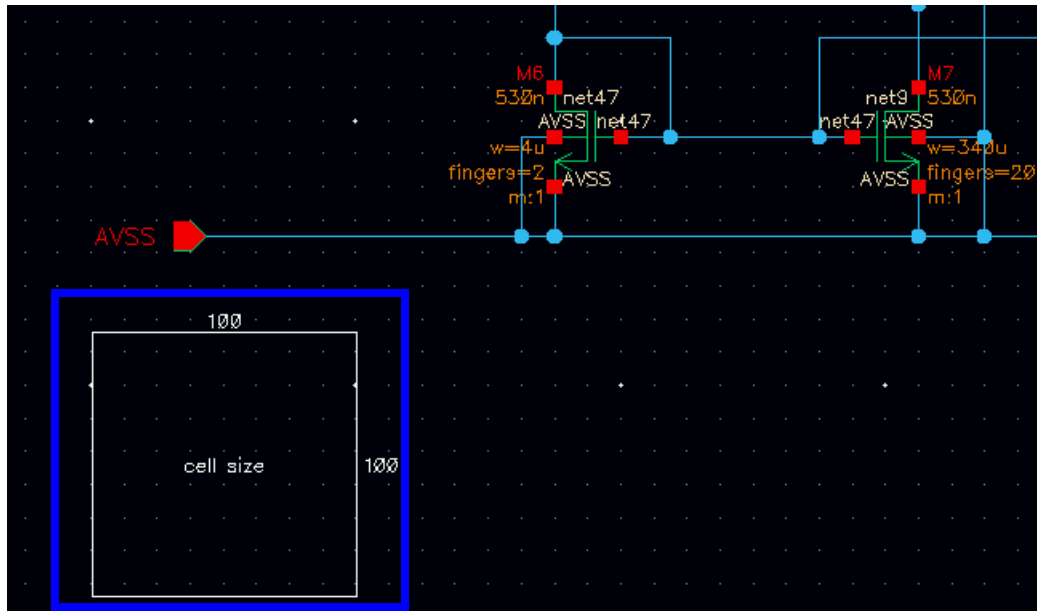
Creating Constraint Notes for a Cell Boundary Constraint

You can only set one Cell Boundary constraint per cellview, with the related constraint note being placed in the top left corner of that cellview.

Once you have created the cellview *Cell Boundary* constraint, either directly in the *Constraint Manager* or using the right-mouse-button contextual menu alternative, a rectangle will be generated to represent the aspect ratio of the max values for the width and height of the cell, and these max values will be printed on each respective side of the rectangle.

Note: This cell boundary constraint note is placed in the bottom left of the schematic.

Figure 1-33 Cell Boundary Constraint Note



User-Defined Constraint Notes for Constraints

The generation of constraint notes can be customized for any constraint type, either to override an existing procedure for note creation, or to add a procedure to create a note for a new constraint type, including custom constraints, if any.

The user-defined constraint notes can be set up for automatic and manual generation as discussed in the following sections above:

- [Setting Up Automatic Generation of Constraint Notes](#)
- [Managing Constraint Notes Manually](#)

Creating a Customized Constraint Note

A new constraint note can be defined using the following user-defined SKILL function:

```
ciCreateCustomNote_<const_type>( cellviewId constId cache )
```

The function takes three arguments. It returns the `dbGroupId` of the note shapes and the generated text.

Make sure that the name of the `dbGroupId` is given in the format "NOTE_<constName>". Also, it must have the following properties:

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

■ conCreateTime

■ conType

The following is an **example of a note generator procedure for an *alignment* constraint**:

```
procedure(ciCreateCustomNote_alignment(cv cx cache)
  let(( instName inst bBox orig orig1 noteGroup memberList shape)
    (foreach member (ciConListMembers cx)
      instName = (car member)
      inst = (dbFindAnyInstByName cv instName) || (car (dbFindMemInstByName
        cv instName))
      (foreach shapes inst~>master~>lpps~>shapes
        when( (car(shapes~>lpp) == '("instance" "drawing"))
          bBox = (getTransformUserBBox (car shapes~>bBox) inst~>transform)
          orig = (list ((xCoord (cadr bBox)) + (xCoord (car bBox)))/2
            ((yCoord (cadr bBox)) + (yCoord (car bBox)))/2)
          ) ; when
        ) ;foreach
      memberList = (append1 memberList orig)
    )

    noteGroup = (dbCreateGroup cv (strcat "NOTE_" (ciConGetName cx)) (list "set"
      "ordered"))
    (when noteGroup
      (dbCreateProp noteGroup "conCreateTime" 'int cx->created->time)
      (dbCreateProp noteGroup "conType" 'string (symbolToString (ciConGetType
        cx)))
    ) ;when

    ;Create a small rectangle at the center of the first member
    orig1 = (car memberList)
    shape = (schCreateNoteShape cv "rectangle" "dashed" (list (list (xCoord
      orig1)-0.03 (yCoord orig1)-0.03) (list (xCoord orig1)+ 0.03
      (yCoord orig1) + 0.03)))
    (dbAddObjectToGroup noteGroup shape)

    ;Create small rectangles at the center of other members and join them to
    the center of the first member
    (foreach orig (cdr memberList)
      shape = (schCreateNoteShape cv "rectangle" "dashed" (list (list (xCoord
        orig)-0.03 (yCoord orig)-0.03) (list (xCoord orig)+ 0.03
        (yCoord orig) + 0.03)))
      (dbAddObjectToGroup noteGroup shape)
```

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

```
        shape = (schCreateNoteShape cv "line" "dashed" (list orig1 orig))
        (dbAddObjectToGroup noteGroup shape)
    ) ;foreach
    noteGroup
) ; let
)
```

Note: When you want constraint notes to be generated automatically, except for a particular type of constraint, a custom note generator could be defined to create an empty note.

The following is an **example of an empty note generator procedure for a *symmetry* constraint:**

```
procedure(ciCreateCustomNote_symmetry(cv cx cache)
  let(( noteGroup )
    noteGroup = dbCreateGroup(cv strcat("NOTE_" ciConGetName(cx)) list("set"
"ordered"))
    when(noteGroup
      dbCreateProp(noteGroup "conCreateTime" 'int ciConGetCreatedTime(cx))
      dbCreateProp(noteGroup "conType" 'string
symbolToString(ciConGetType(cx)))
    ) ;when
    noteGroup
  ) ; let
)
```

User-Defined Constraint Notes for Templates

To generate user-defined constraint notes for a template, create a SKILL function, `ciTemplateCreateCustomNote_<templateType>` that has the following syntax:

```
ciTemplateCreateCustomNote_<templateType>( cellviewId templateId cacheId )
```

The function takes three arguments. It returns the `dbGroupId` that consists of all the geometries, such as, the note shapes and the generated text.

Make sure that the name of the `dbGroupId` is given in the format `TemplateNote_<templateName>`. Also, it must have the following properties:

- `templateCreateTime` (This value can be set using `ciTemplateGetCreatedTime(templateId)`)
- `templateType` (This value can be set using `ciTemplateGetType(templateId)`)

The user-defined constraint notes for templates can be set up for automatic and manual generation as discussed in the following sections above:

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

- Setting Up Automatic Generation of Constraint Notes
- Managing Constraint Notes Manually

By default, constraint notes are created for templates and their member constraints. To change this behavior, you can create a SKILL function, `ciTemplateNoteControl_<templateType>`, for each template type using the following syntax:

```
ciTemplateNoteControl_<templateType> (cellView templateId cacheId)
```

Here, the `templateId` argument allows you to create constraint notes for different templates of same type in different ways.

The SKILL function should return one of the following four values:

<code>`templateNotesOnly</code>	Notes are created for the specified template only. Any constraint notes present on member constraints are removed.
<code>`constraintNotesOnly</code>	Notes are created only for member constraints that are eligible for creation of constraint notes. Constraint note is not created on the specified template. Any constraint note present on the template is removed.
<code>`templateAndConstraintNotes</code>	Notes are created for the specified template and its member constraints that are eligible for creation of constraint notes. This is the default return value.
<code>`noTemplateOrConstraintNotes</code>	Notes are not created for the specified template or its member constraints. Any constraint notes present on the template or its member constraints are removed.

Note: If the return value of the SKILL function, `ciTemplateNoteControl_<templateType>`, is ``templateNotesOnly` or ``noTemplateOrConstraintNotes`, constraint notes cannot be created on the member constraints of the specified template. If this SKILL function returns an invalid symbol, a warning is displayed in the CIW when you try to create a constraint note on a template of that particular type and the default value ``templateAndConstraintNotes` will be used.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Example of Creating User-Defined Constraint Notes for a Template

The following is an **example of a note generator procedure for a *Matching (strength)* template type**:

```
procedure(ciTemplateCreateCustomNote_match(cv cx cache)
  let(( instName inst bBox orig origl noteGroup memberList shape label myBBox
    minXL minYL maxXL maxYL maxX minX maxY minY tempL)
    (foreach member (ciConListMembers (car ciTemplateListCon(cx)))
      instName = (car member)
      inst = (dbFindAnyInstByName cv instName) || (car (dbFindMemInstByName
        cv instName))
      (foreach shapes inst~>master~>lpps~>shapes
        when( (car(shapes~>lpp) == '("instance" "drawing"))
          bBox = (getTransformUserBBox (car shapes~>bBox)
            inst~>transform)
          orig = (list ((xCoord (cadr bBox)) + (xCoord (car bBox)))/2
            ((yCoord (cadr bBox)) + (yCoord (car bBox)))/2)
          when( bBox
            minXL = cons( xCoord(xCoord(bBox)) minXL )
            maxXL = cons( xCoord(yCoord(bBox)) maxXL )
            minYL = cons( yCoord(xCoord(bBox)) minYL )
            maxYL = cons( yCoord(yCoord(bBox)) maxYL )
          ) ; when
        ) ; when
      ) ; foreach

      memberList = (append1 memberList orig)
    )

    tempL = minXL
    minX = car(sort(tempL 'lessp))
    tempL = maxXL
    maxX = car(sort(tempL 'greaterp))
    tempL = minYL
    minY = car(sort(tempL 'lessp))
    tempL = maxYL
    maxY = car(sort(tempL 'greaterp))

    ;myBBox = list(list(minX-0.1 minY-0.1) list(maxX+0.1 maxY+0.1))
    myBBox = list(list(minX-0.1 minY-0.1) list(minX-0.5 minY+((maxY-minY)/2))
      list(minX-0.1 maxY-0.1))
```

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

```
list(maxX+0.1 maxY+0.1) list(maxX+0.5 minY+((maxY-minY)/2))
list(maxX+0.1 minY+0.1))

noteGroup = (dbCreateGroup cv (strcat "TemplateNote_" (ciTemplateGetName
cx)) (list "set" "ordered"))
(when noteGroup
  (dbCreateProp noteGroup "templateCreateTime" 'int
(ciTemplateGetCreatedTime cx))
  (dbCreateProp noteGroup "templateType" 'string (ciTemplateGetType cx))
) ;when

shape = (schCreateNoteShape cv "polygon" "solid" myBBox)
;shape = (schCreateNoteShape cv "rectangle" "dashed" myBBox)
(dbAddObjectToGroup noteGroup shape)
label = (schCreateNoteLabel cv list(maxX - 0.05 maxY+0.15)
(ciTemplateGetType cx) "centerCenter" "R0" "stick"
_constraintNotesGlobals.__FONT_SIZE__ "normalLabel")
(dbAddObjectToGroup noteGroup label)

noteGroup
) ; let
)

procedure(ciTemplateNoteControl_match(cv tplate cache)
  let((ret)
    ret = 'templateAndConstraintNotes
    when(tplate
      case(tplate->name
        ("MatchTemplate1" ret = 'templateNotesOnly)
        ("MatchTemplate2" ret = 'constraintNotesOnly)
        (t ret = 'templateAndConstraintNotes)
      )
    )
    ret
  )
)
```

Registering Pre- and Post-Create Triggers for Constraint Notes

By default, any modifications made to the position of a constraint note for a constraint or a template do not reflect after the constraint note is updated. If you want to restore the modified

position, write the pre-create trigger to store the position and the post-create trigger to restore it.

The following SKILL functions, when written for each constraint type, enable you to register the pre- and post-create triggers:

- `ciCreatePreProcNote_<const_type>( constId dbGroupId )`
- `ciCreatePostProcNote_<const_type>( constId dbGroupId )`

For example, to create a pre-create trigger for a `symmetry` constraint note, write a SKILL function as following:

```
ciCreatePreProcNote_symmetry ( constId dbGroupId )
```

Similarly, you can write the following SKILL functions for each template type to register the pre- and post-create triggers:

- `ciCreatePreProcTemplateNote_<templateType>(templateId noteGroup)`
- `ciCreatePostProcTemplateNote_<templateType>(templateId noteGroup)`

Note: A pre-create procedure is called only if there is an existing constraint note on the specified constraint or template. While a post-create procedure is called only if the constraint note is successfully created.

In addition to providing the store and restore ability, the pre- and post-create triggers allow you to define any processing requirements to be executed before and after updating a constraint note.

Creating Constraint Notes for a Complete Hierarchy

Constraint notes can be created in all schematic views from the top cellview down using the following SKILL function:

```
ciAddHierarchicalNotes( t_libName t_cellName t_viewName [ g_hierarchical ] )
```

For example:

```
ciAddHierarchicalNotes( "designLib" "topCell" "schematic" t )
```

For more information, see [ciAddHierarchicalNotes](#).

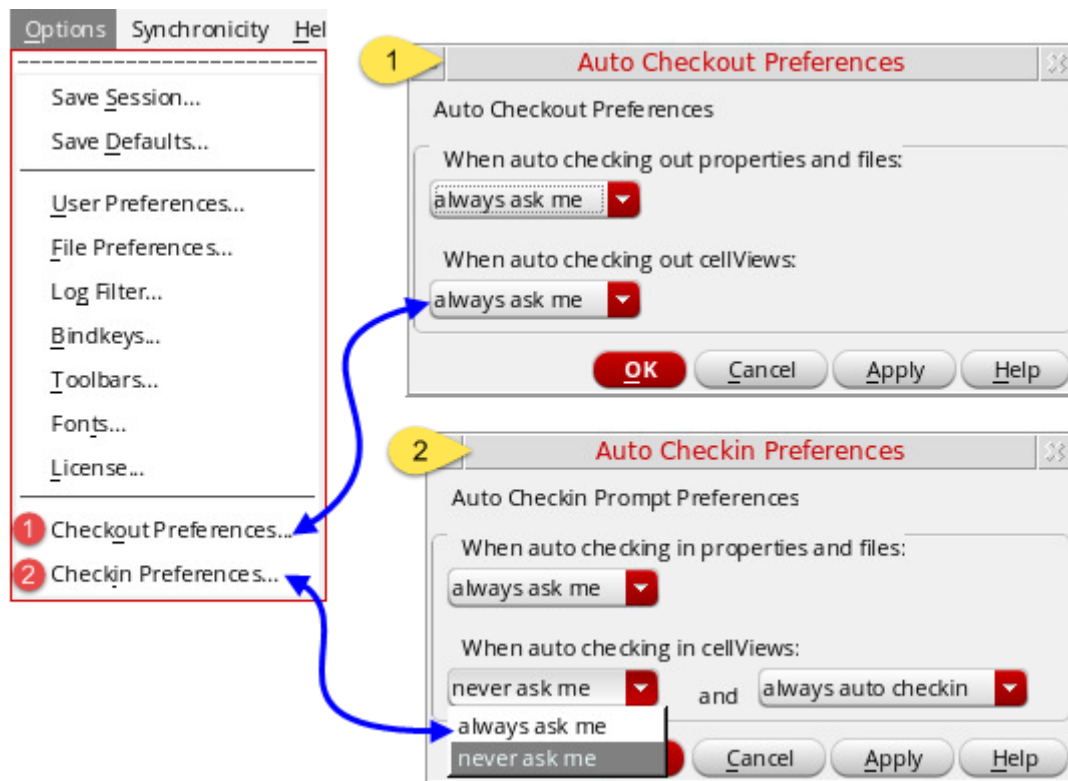
Working with Constraint Views in a Design-Managed Environment

A design-managed (DM) environment uses Cadence Generic Design Management (GDM) or a third-party design management tool like Synchronicity DesignSync (DSS) for design data version control and management. When you work with constraint views in a DM environment, operations such as save, discard edit, close data, and make read-only are monitored closely to determine check out, check in, and save requirements for version management. The behavior is consistent for these operations across the layout, schematic, and constraint views in a DM environment.



Tip

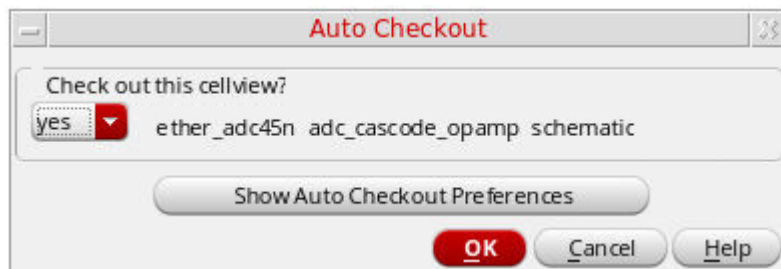
To view prompts that allow you to control the check in and check out of cellviews and constraint views in a DM environment, access the *Options* menu in the CIW and set *Checkout Preferences* and *Checkin Preferences* to *always ask me*, as shown below.



Virtuoso Unified Custom Constraints User Guide

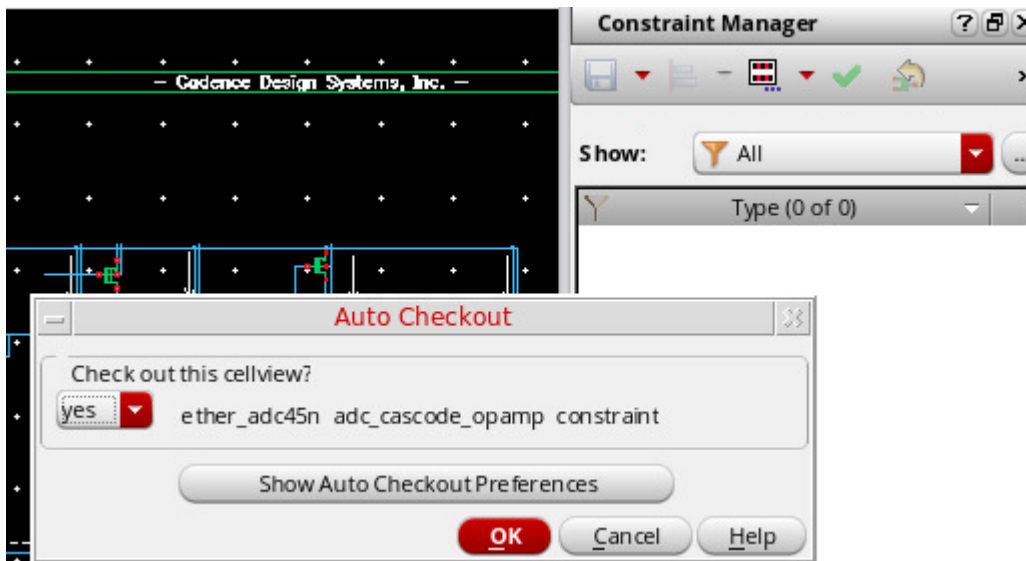
The Constraint Manager Assistant

When these preferences are set in a DM environment and you check out a managed cellview or constraint view, the confirmation form, *Auto Checkout*, is displayed, as shown below:



Note: The *Show Auto Checkout Preferences* button in the *Auto Checkout* form can be used to reset your checkout preferences.

When you open Schematic XL to view a checked out schematic view, you are prompted to check out the associated constraint view, if any, as shown below:



The table below lists different scenarios related to constraint views and their corresponding expected behavior in DM environment:

Scenario	Expected Behavior
1. Purging constraint view using the <i>File – Close Data</i> menu in the CIW	
■ Purge the constraint view without making changes to it.	You are prompted to check in the constraint view.

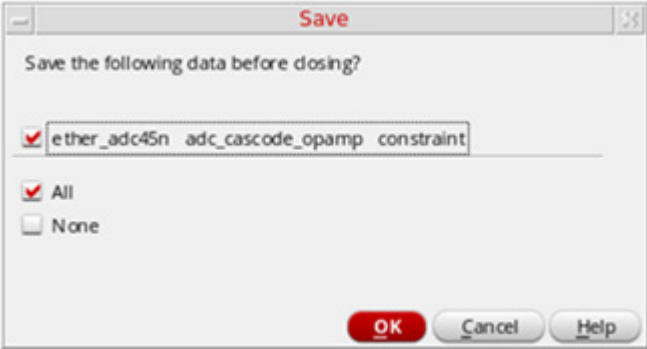
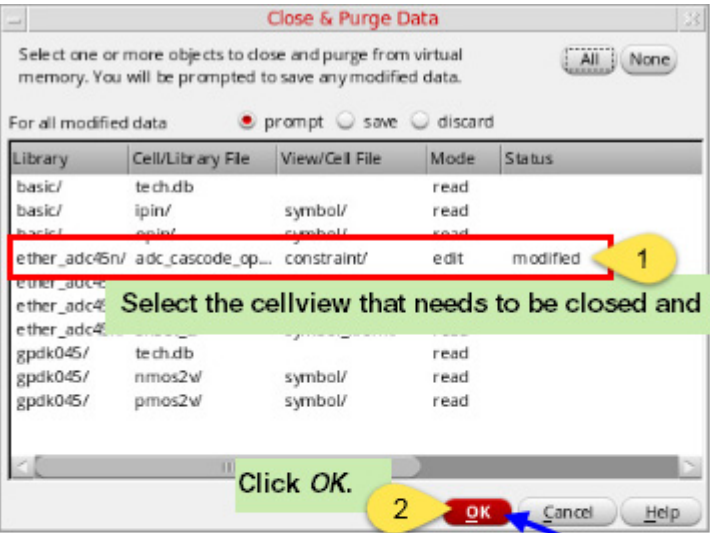
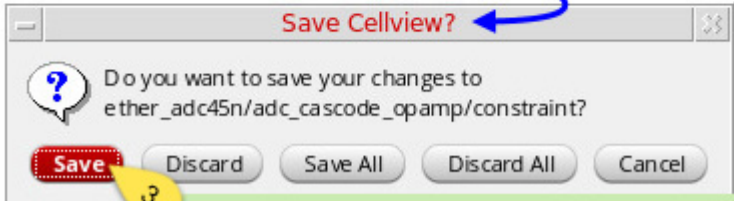
Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Scenario	Expected Behavior
■ Purge a constraint view after saving the changes that you made to it.	When you close and purge the constraint view, you are prompted to check in the selected constraint view, as shown below: In the <i>Auto Checkin</i> form, you can choose to check in the constraint view (<i>yes</i>), leave it checked out (<i>no</i>), or cancel the checkout operation (<i>cancel checkout</i>).  Tip Use the <i>Show Auto Checkin Preferences</i> button on the <i>Auto Checkin</i> form to reset your check-in preferences.

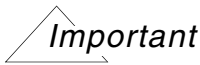
Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Scenario	Expected Behavior
■ Purge a constraint view without saving the changes that you made to it.	When you exit Schematic XL, you are prompted to save the updated constraint view, as shown below:  If you click <i>Cancel</i> on the Save form and exit Schematic XL, then on purging the constraint view, you are first prompted to save the selected constraint view as shown below:  Select the cellview that needs to be closed and purged. Click OK.  Select <i>Save</i> to save your changes to the selected cellview.

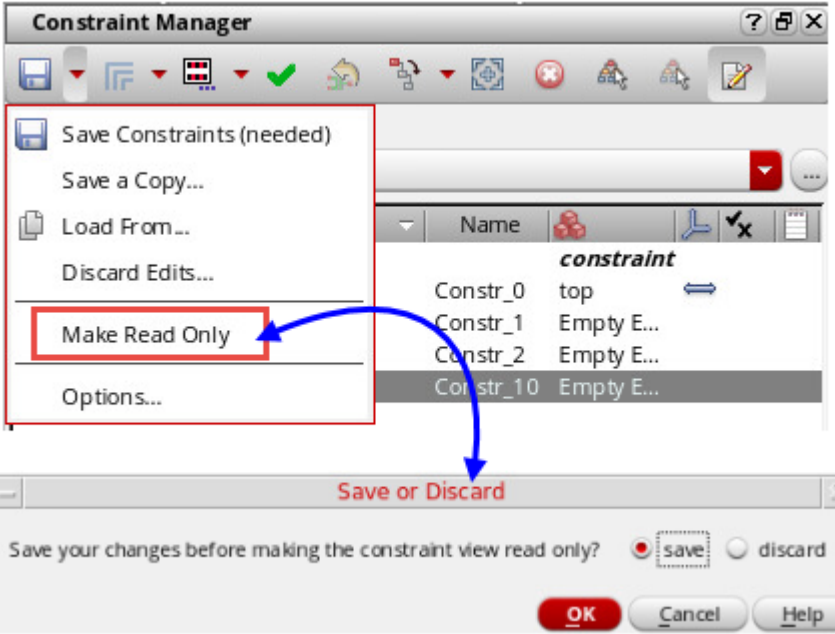
Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Scenario	Expected Behavior
	After the constraint view is saved successfully, you are prompted to check it in. Note: You are prompted to save and check in the constraint view if you exit Virtuoso without saving a modified constraint view or when you perform a close data operation on it.
2. Closing the schematic cellview	
■ Modify the schematic and constraint view, and close Schematic XL after saving the updates.	You are prompted to check in the modified and saved schematic and constraint views.
■ Modify the schematic and constraint view, and close Schematic XL without saving the updates.	You are prompted to choose whether you want to save or discard the modifications made to the schematic and constraint views. Regardless of the choice you make, you are prompted to confirm whether the schematic and constraint views need to be checked in.
 Important When you open or close a schematic view that does not have an associated constraint view in Schematic XL, you are not prompted to check in the constraint view.	

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Scenario	Expected Behavior
3. Making the constraint view read-only	
■ Make the constraint view read-only by choosing <i>File – Make Read Only</i> in Schematic XL or in the Constraint Manager assistant.	The <i>Save or Discard</i> form, shown below, prompts you to save or discard the changes made to the constraint view before the constraint view becomes read-only.  The screenshot shows the 'Constraint Manager' window. A red box highlights the 'File' menu, and a blue arrow points from the 'Make Read Only' option to the 'Save or Discard' dialog box. The dialog box has the title 'Save or Discard' and the text 'Save your changes before making the constraint view read only?'. It has two radio buttons: 'save' (selected) and 'discard'. At the bottom are 'OK', 'Cancel', and 'Help' buttons. Choose <i>save</i> or <i>discard</i>, and click <i>OK</i>. You are next prompted to check in the constraint view. Note: You can switch the constraint view to edit mode by using <i>File – Make Editable</i> in Schematic XL or in the Constraint Manager assistant. As a result, you are prompted to check out the constraint view.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Scenario	Expected Behavior																												
4. Saving the constraint view																													
Save the constraint view by choosing <i>File – Save (Constraint)</i> in Schematic XL or <i>File – Save Constraints (needed)</i> in the Constraint Manager assistant.	The constraint view is saved and you are not prompted to check in the constraint view. Notice that the checkout and lock status of the constraint view remain unchanged as shown in the Library Manager below: View Filter: Ctrl-Enter to filter schematic <table><thead><tr><th>View</th><th>Lock</th><th>DM state</th><th>Size</th></tr></thead><tbody><tr><td>✓ Doc_Specs</td><td></td><td>checked in</td><td>6k</td></tr><tr><td>✓ abstract</td><td></td><td>checked in</td><td>41k</td></tr><tr><td>✓ av_extracted1</td><td></td><td>checked in</td><td>379k</td></tr><tr><td>✓ av_extracted2</td><td></td><td>checked in</td><td>394k</td></tr><tr><td>🔒 constraint</td><td></td><td>checked out: <i>user</i></td><td>32</td></tr><tr><td>✓ constraint1</td><td></td><td>checked in</td><td>32</td></tr></tbody></table>	View	Lock	DM state	Size	✓ Doc_Specs		checked in	6k	✓ abstract		checked in	41k	✓ av_extracted1		checked in	379k	✓ av_extracted2		checked in	394k	🔒 constraint		checked out: <i>user</i>	32	✓ constraint1		checked in	32
View	Lock	DM state	Size																										
✓ Doc_Specs		checked in	6k																										
✓ abstract		checked in	41k																										
✓ av_extracted1		checked in	379k																										
✓ av_extracted2		checked in	394k																										
🔒 constraint		checked out: <i>user</i>	32																										
✓ constraint1		checked in	32																										
5. Discarding edits made in the constraint view																													
Discard the edits made in the constraint view by choosing <i>File – Discard Edits (Constraint)</i> in Schematic XL or <i>File – Discard Edits</i> in the Constraint Manager assistant.	A form is displayed only to check whether you want to discard the unsaved modifications in the constraint view. You are not prompted to check in the constraint view. The checkout and lock status of the constraint view remain unchanged.																												

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Scenario	Expected Behavior
6. Opening schematic view with no associated pre-existing constraint view	
■ Open Schematic XL and close the schematic view without creating any constraints.	You are not prompted to check out a constraint view and a constraint directory is not created in the workspace because a constraint view does not exist. Also, in the Library Manager, no constraint view is displayed in the <i>View</i> section for the selected <i>Library – Cell</i> combination. Note: The Constraint Manager assistant is available for creation of new constraints even when there is no pre-existing constraint view. If you close the schematic view without creating any constraints, you are not prompted to check in the constraint view because none was created.

Constraint Manager in Layout XL

You can use the Constraint Manager in Virtuoso® Layout XL to add, modify, check or delete constraints in your design.

Examples of constraints include matching parameters, symmetry, orientation, matched orientation, IR drop, parasitic filtering, parasitic estimation, clustering, alignment, area, distance, boundaries, power structure/guard rings (layout only), custom module generators (modgens), cell plans, and analog groups. For a complete list of the constraints supported by Layout XL, see Constraints in Virtuoso Schematic and Layout Editing Environment in the *Virtuoso Unified Custom Constraints User Guide*.

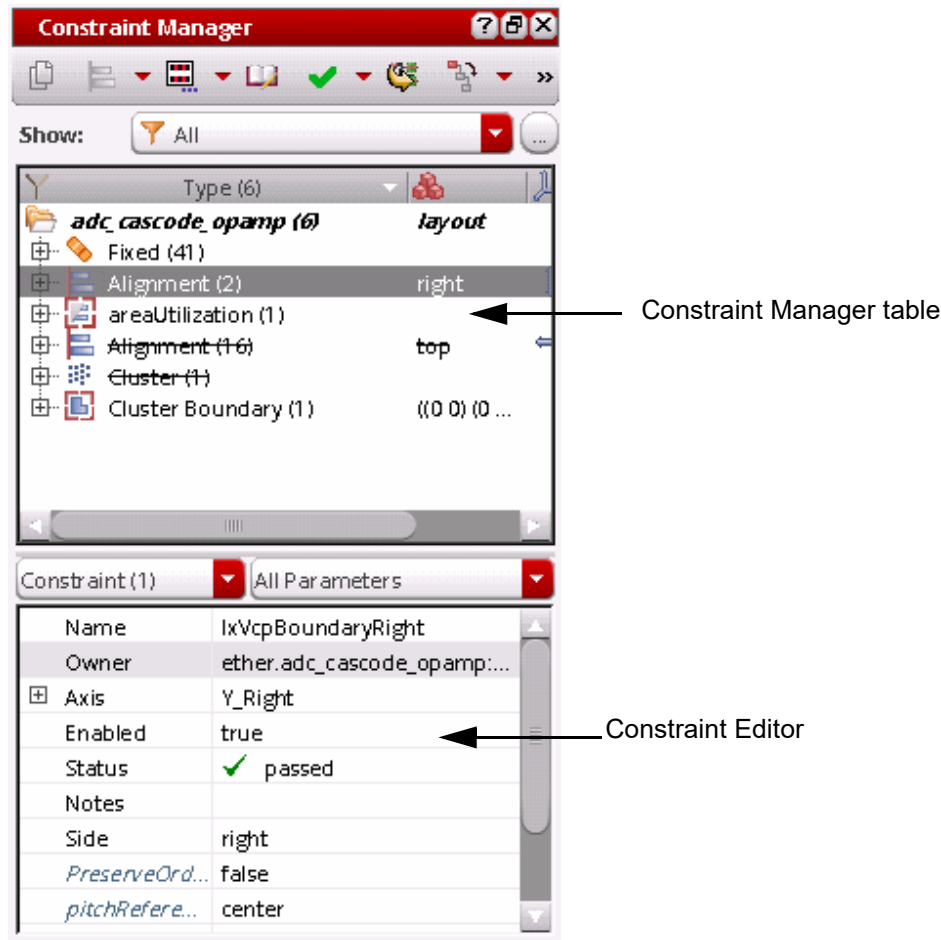
The Constraint Manager displays a full set of constraints for a design wherever you are in the design hierarchy and wherever the constraints were created in that hierarchy. It displays the constraints in a logical manner, and shows which constraints are currently met and which have been overridden during the course of the physical implementation.

The Constraint Manager user interface comprises two main component parts; the Constraint Manager table at the top, which lets you browse the constraints in your design; and the

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Constraint Editor underneath it, which lets you change the values of one or more selected constraints.



For more information on the Constraint Manager toolbar, see [Constraint Manager Toolbar](#).

The Constraint Manager has a dedicated Batch Checker in the toolbar, which automatically enables the placement, specialty routing, and process constraints. In addition, the checker verifies and updates the constraint status and leaves them back as is in the 'Verify Design' form.

To run other checks, you should invoke *Verify Design* and click OK after enabling the required check.

Related Topics

[Constraint Manager Assistant](#)

[Updating Constraints From Schematic to Layout](#)

[Comparing Constraints between Schematic and Layout](#)

[Overriding Process Rules](#)

[Creating a Cluster Boundary Using the Constraint Manager](#)

[Creating and Editing Net Constraints](#)

Updating Constraints From Schematic to Layout

To update constraints in the layout to match those set in the schematic:

- In the Constraint Manager toolbar, click the *Update Layout Constraints* button.



Constraints in the schematic are transferred to the top-level layout view. Constraints that have been created in the schematic but not yet saved are also transferred.

For more information, see [Constraint Transfer](#).

Related Topics

[Constraint Manager in Layout XL](#)

[Comparing Constraints between Schematic and Layout](#)

[Overriding Process Rules](#)

[Creating a Cluster Boundary Using the Constraint Manager](#)

[Creating and Editing Net Constraints](#)

Comparing Constraints between Schematic and Layout

When you click the Constraint Comparison Report () button in the Constraint Manager assistant toolbar of Layout XL, the Constraint Comparison Report opens in the (default) web browser. The report contains a comparison of constraints in the currently open Schematic XL and Layout XL views.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Starting from IC6.1.6 ISR9 and ICADV12.1 ISR11, the Constraint Comparison Report is displayed in the format shown in the figure below:

						Last save date	Status	Revision
Constraint Comparison Report						Fri Sep 19 09:15:09 2014		
ether_adc45n/adc_sample_hold:schematic						Mon May 16 15:45:01 2011	Not Modified	Not Managed
ether_adc45n/adc_sample_hold:layout_routed1Test						Mon Sep 1 16:27:55 2014	Not Modified	Not Managed
ether_adc45n.adc_sample_hold:constraint						Tue Jun 12 14:41:46 2012	Modified	Not Managed

Status	Constraint Type	+	=	#	Name	Type	ether_adc45n adc_sample_hold schematic	ether_adc45n adc_sample_hold layout_routed1Tes
Violated	Distance	+	=				Distance.00	_Distance.0
Violated and Not Bound	Alignment	+		#				lxVcpBoundaryTop
Violated and Not Bound	Matched Length	+		#				Constr_9
Violated and Not Bound	Orientation	+		#				_Constr_32
Violated and Not Bound	Symmetry	+		#				Constr_2
Mismatch	Guard Ring	+	=	#			_GuardRing_Modgen_3	_GuardRing_Modgen_
Mismatch	Guard Ring	+	=	#			_GuardRing_Modgen_4	_GuardRing_Modgen_
Mismatch	Modgen	+	=	#			_Constr_8	_Constr_8
Mismatch	Modgen	+	=	#			_Modgen_1	_Constr_21
Mismatch	Process Rule Overrides	+	=	#			CG_25	CG_72
Mismatch	Symmetry	+	=	#			_Constr_23	_Constr_23
Mismatch	Symmetry	+	=	#			_Constr_24	_Constr_24
Mismatch	Symmetry	+	=	#			_Constr_28	_Constr_28
Mismatch	Symmetry	+	=	#			_Constr_29	_Constr_29

Note: If you prefer the constraint comparison report format that used to be displayed until IC6.1.6 ISR8 and ICADV12.1 ISR10, set the `useOldComparisonReport` environment variable to `t`.

The associated HTML file, css file, and JavaScript code are saved in the current working directory, as shown below:

```
/ConstraintReports/<libName>/<cellName>/<viewName>/<DATE_AS_YEAR_MONTH_DAY_TIME>/
```

The Constraint Comparison Report begins with a summary of the session, as shown below. The first column lists the names of cellviews compared. The next column displays the date on which each cellview was last saved and its current status, that is, *Modified* or *Not Modified*. The *Revision* column displays the revision number if the design view is managed

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

by a revision system supported by Generic Design Management (GDM). If the revision number cannot be retrieved, *Not Managed* is displayed in the last column.

	Last save date	Status	Revision
Constraint Comparison Report	Fri Sep 19 09:15:09 2014		
ether_adc45n/adc_sample_hold:schematic	Mon May 16 15:45:01 2011	Not Modified	Not Managed
ether_adc45n/adc_sample_hold:layout_routed1Test	Mon Sep 1 16:27:55 2014	Not Modified	Not Managed
ether_adc45n.adc_sample_hold:constraint	Tue Jun 12 14:41:46 2012	Modified	Not Managed

The summary table is followed by another table in which each constraint in the schematic and layout cellviews is compared. In the header row of this table, each cell is clickable. The arrangement of the displayed comparison information changes based on whether you click

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

the column header once or twice. The impact of the different types of clicks on a clickable column header is described in the tables below.

Column Name	Action: Click	Action: Double-click
Status	Orders the content in the following sequence: ■ <i>Status</i> column is arranged from lowest to highest severity. ■ <i>Constraint Type</i> column is arranged in descending alphabetical order. ■ <i>Name</i> column is arranged in descending alphabetical order.	Reverses the sort order applied by a single click.
Constraint Type	Orders the content in the following sequence: ■ <i>Constraint Type</i> column is arranged in descending alphabetical order. ■ <i>Status</i> column is arranged from lowest to highest severity. ■ <i>Name</i> column is arranged in descending alphabetical order.	Reverses the sort order applied by a single click.
[...] schematic	Orders the content in the following sequence: ■ Schematic constraint names listed in the [...] <i>schematic</i> column are arranged in descending alphabetical order. ■ <i>Status</i> column is arranged from lowest to highest severity. ■ <i>Constraint Type</i> column is arranged in descending alphabetical order. ■ <i>Name</i> column is arranged in descending alphabetical order.	Reverses the sort order applied by a single click.
Note: A cell with black color fill displayed in this column means that the constraint exists in the layout, but not in the schematic. A cell with gray color fill means that the constraint is disabled or out of context.		

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Column Name	Action: Click	Action: Double-click
[...] layout	Orders the content in the following sequence: ■ Layout constraint names listed in the [...] <i>layout</i> column are arranged in descending alphabetical order. ■ <i>Status</i> column is arranged from lowest to highest severity. ■ <i>Constraint Type</i> column is arranged in descending alphabetical order. ■ <i>Name</i> column in arranged in descending alphabetical order.	Reverses the sort order applied by a single click.
	Note: A cell with black color fill displayed in this column means that the constraint exists in the schematic, but not in the layout. A cell with gray color fill means that the constraint is disabled or out of context.	

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Column Name	Action: Click																																																																																																																																																																																										
+	The first click expands all constraints to display information about the associated constraint parameters and members. The next click collapses all expanded constraints simultaneously. When you click the + column header, the label changes to - (minus). In addition, the + button in each row changes to -. You can click these buttons to display or collapse the information related to the constraint compared in that row. If you click the - button corresponding to a constraint, information related to only that constraint is collapsed. Click to expand or collapse all constraints. <table><tr><th>Status</th><th>Constraint Type</th><th>-</th><th>=</th><th>#</th><th>Name</th><th>Type</th><th>ether_adc45n adc_sample_hold schematic</th><th>ether_adc45n adc_sample_hold layout_routed1Test</th></tr><tr><td rowspan="19">Violated</td><td>Distance</td><td>-</td><td>-</td><td></td><td></td><td></td><td> Distance.00</td><td>_ Distance.0</td></tr><tr><td></td><td></td><td></td><td></td><td>Notes</td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td>Axis</td><td></td><td>bi-directional (default)</td><td>bi-directional (default)</td></tr><tr><td></td><td></td><td></td><td></td><td>direction</td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td>axisLocation</td><td></td><td>any</td><td>any</td></tr><tr><td></td><td></td><td></td><td></td><td>coordinate</td><td></td><td>0</td><td>0</td></tr><tr><td></td><td></td><td></td><td></td><td>realizedCoordinate</td><td></td><td>0</td><td>0</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="2">Parameters</td></tr><tr><td></td><td></td><td></td><td></td><td>Min</td><td></td><td>0.25</td><td>0.25</td></tr><tr><td></td><td></td><td></td><td></td><td>Max</td><td></td><td>(not set)</td><td>(not set)</td></tr><tr><td></td><td></td><td></td><td></td><td>reference</td><td></td><td>edge</td><td>edge</td></tr><tr><td></td><td></td><td></td><td></td><td>use</td><td></td><td>boundary</td><td>boundary</td></tr><tr><td></td><td></td><td></td><td></td><td>layerName</td><td></td><td>OVERLAP</td><td>OVERLAP</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="2">Members</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td>inst</td><td>NM10</td><td>NM10</td></tr><tr><td></td><td></td><td></td><td></td><td>reference</td><td></td><td>edge</td><td>edge</td></tr><tr><td></td><td></td><td></td><td></td><td>use</td><td></td><td>boundary</td><td>boundary</td></tr><tr><td></td><td></td><td></td><td></td><td>layerName</td><td></td><td>OVERLAP</td><td>OVERLAP</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td>inst</td><td>NM9</td><td>NM9</td></tr><tr><td></td><td></td><td></td><td></td><td>reference</td><td></td><td>edge</td><td>edge</td></tr><tr><td></td><td></td><td></td><td></td><td>use</td><td></td><td>boundary</td><td>boundary</td></tr><tr><td></td><td></td><td></td><td></td><td>layerName</td><td></td><td>OVERLAP</td><td>OVERLAP</td></tr></table>	Status	Constraint Type	-	=	#	Name	Type	ether_adc45n adc_sample_hold schematic	ether_adc45n adc_sample_hold layout_routed1Test	Violated	Distance	-	-				Distance.00	_ Distance.0					Notes								Axis		bi-directional (default)	bi-directional (default)					direction								axisLocation		any	any					coordinate		0	0					realizedCoordinate		0	0							Parameters						Min		0.25	0.25					Max		(not set)	(not set)					reference		edge	edge					use		boundary	boundary					layerName		OVERLAP	OVERLAP							Members							inst	NM10	NM10					reference		edge	edge					use		boundary	boundary					layerName		OVERLAP	OVERLAP						inst	NM9	NM9					reference		edge	edge					use		boundary	boundary					layerName		OVERLAP	OVERLAP
Status	Constraint Type	-	=	#	Name	Type	ether_adc45n adc_sample_hold schematic	ether_adc45n adc_sample_hold layout_routed1Test																																																																																																																																																																																			
Violated	Distance	-	-				Distance.00	_ Distance.0																																																																																																																																																																																			
					Notes																																																																																																																																																																																						
					Axis		bi-directional (default)	bi-directional (default)																																																																																																																																																																																			
					direction																																																																																																																																																																																						
					axisLocation		any	any																																																																																																																																																																																			
					coordinate		0	0																																																																																																																																																																																			
					realizedCoordinate		0	0																																																																																																																																																																																			
							Parameters																																																																																																																																																																																				
					Min		0.25	0.25																																																																																																																																																																																			
					Max		(not set)	(not set)																																																																																																																																																																																			
					reference		edge	edge																																																																																																																																																																																			
					use		boundary	boundary																																																																																																																																																																																			
					layerName		OVERLAP	OVERLAP																																																																																																																																																																																			
							Members																																																																																																																																																																																				
						inst	NM10	NM10																																																																																																																																																																																			
					reference		edge	edge																																																																																																																																																																																			
					use		boundary	boundary																																																																																																																																																																																			
					layerName		OVERLAP	OVERLAP																																																																																																																																																																																			
						inst	NM9	NM9																																																																																																																																																																																			
				reference		edge	edge																																																																																																																																																																																				
				use		boundary	boundary																																																																																																																																																																																				
				layerName		OVERLAP	OVERLAP																																																																																																																																																																																				

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

Column Name	Action: Click
=	The first click expands all constraints that have the same constraint parameters and members to display the corresponding information. The next click collapses such constraints simultaneously. When you click the = column header, the label changes to - (minus). In addition, the  button in each row containing constraints with same constraint parameters and members changes to -. You can click these buttons to display or collapse the information related to the constraint compared in that row. If you click the - button corresponding to a constraint, information related to only that constraint is collapsed.
#	The first click expands all constraints that have different constraint parameters and members to display the corresponding information. The next click collapses such constraints simultaneously. When you click the # column header, the label changes to - (minus). In addition, the # button in each row containing constraints that have the different constraint parameters and members changes to -. You can click these buttons to display or collapse the information related to the constraint compared in a specific row. If you click the - button corresponding to a constraint, the information related to only that constraint is collapsed.

Video

For a video demonstration of the process of generating a constraint comparison report, see [Comparing Constraints Between Schematic and Layout](#).

Related Topics

[Constraint Manager in Layout XL](#)

[Updating Constraints From Schematic to Layout](#)

[Overriding Process Rules](#)

[Creating a Cluster Boundary Using the Constraint Manager](#)

[Creating and Editing Net Constraints](#)

Overriding Process Rules

To launch the Process Rule Editor,

- In the Constraint Manager toolbar, click the *Process Rule Editor* button.



Related Topics

[Constraint Manager in Layout XL](#)

[Updating Constraints From Schematic to Layout](#)

[Comparing Constraints between Schematic and Layout](#)

[Creating a Cluster Boundary Using the Constraint Manager](#)

[Creating and Editing Net Constraints](#)

Creating a Cluster Boundary Using the Constraint Manager

To create a cluster boundary constraint using the Constraint Manager graphical user interface:

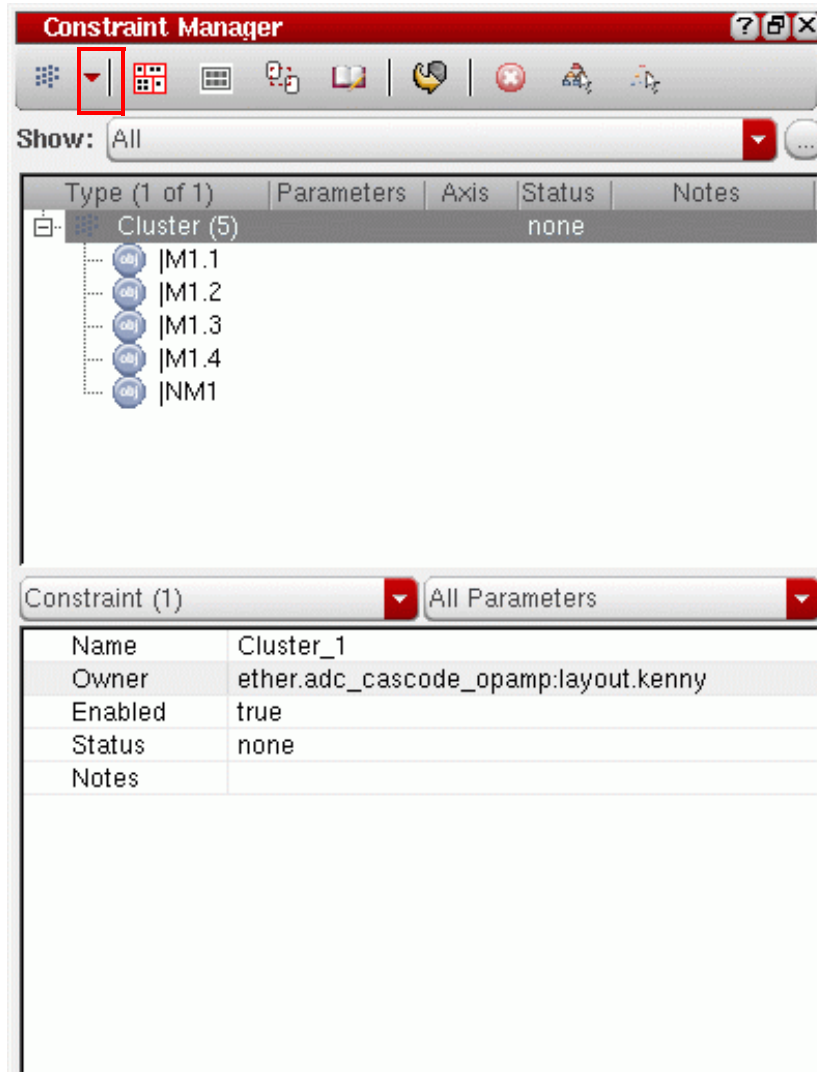
1. In the Constraint Manager assistant, select an existing cluster around which you want to create a cluster boundary.

Note: You can create a cluster boundary only for an existing cluster.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

2. Choose *Cluster Boundary* from the constraint creation pull-down.

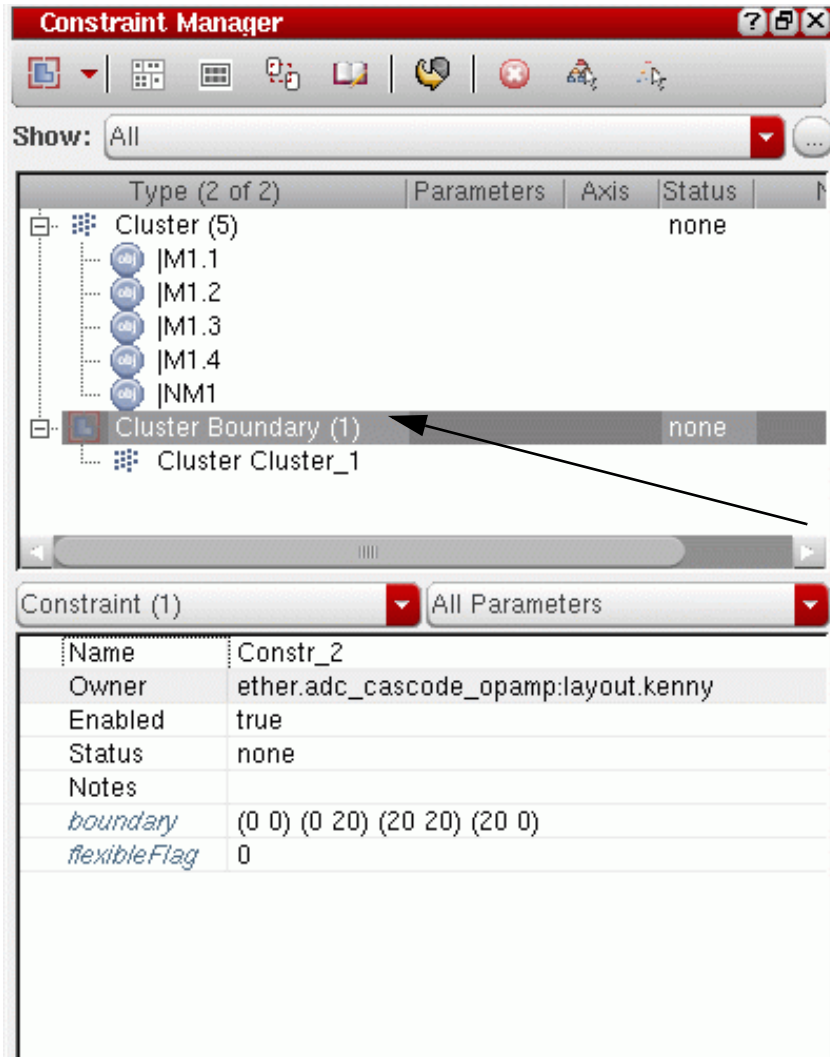


A cluster boundary constraint is created with the selected cluster constraint as a member. The cluster boundary is created with a default size. You are not prompted to draw the cluster boundary on the layout canvas.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

3. Specify the size for the cluster boundary by typing the required coordinates into the appropriate fields in the Constraint Manager assistant.



Specify the size of the cluster

Related Topics

[Constraint Manager in Layout XL](#)

[Updating Constraints From Schematic to Layout](#)

[Comparing Constraints between Schematic and Layout](#)

[Overriding Process Rules](#)

[Creating and Editing Net Constraints](#)

Creating and Editing Net Constraints

You can use a combination of the navigator assistant and the Constraint Manager assistant to create and edit *Net Priority* and *Net Class* constraints on existing nets in your design.

Creating a Net Priority Constraint

A *Net Priority* constraint is an ordered net constraint which defines the order of priority when routing a net. A higher priority implies that the net is more critical and therefore needs to be routed more optimally (in terms of wire length and any other user-defined constraint on it), even if that makes routing of the lower priority nets less optimal.

Net Priority constraints are honored by the Virtuoso Floorplanner, Virtuoso Analog Placer, and the Virtuoso Space-based Router.

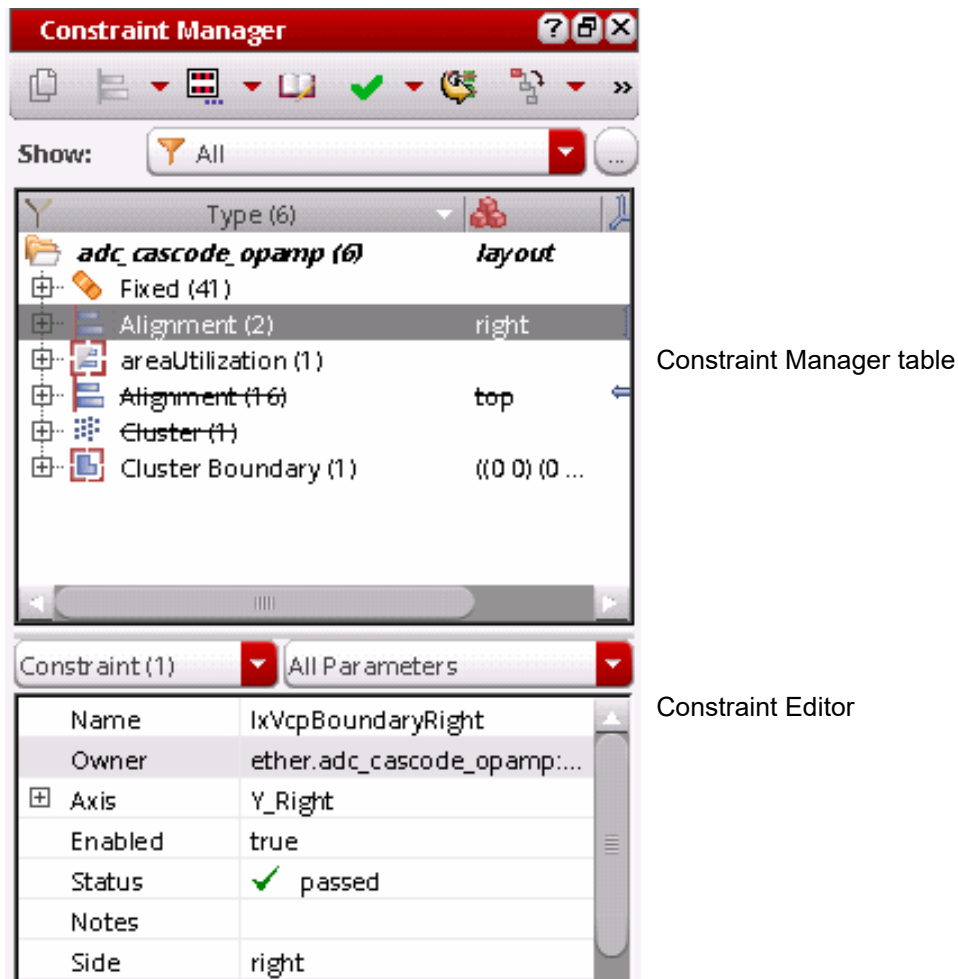
To create a *Net Priority* constraint,

1. From the layout window menu bar, choose *Windows – Assistants – Constraint Manager*.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

The Constraint Manager assistant is displayed.

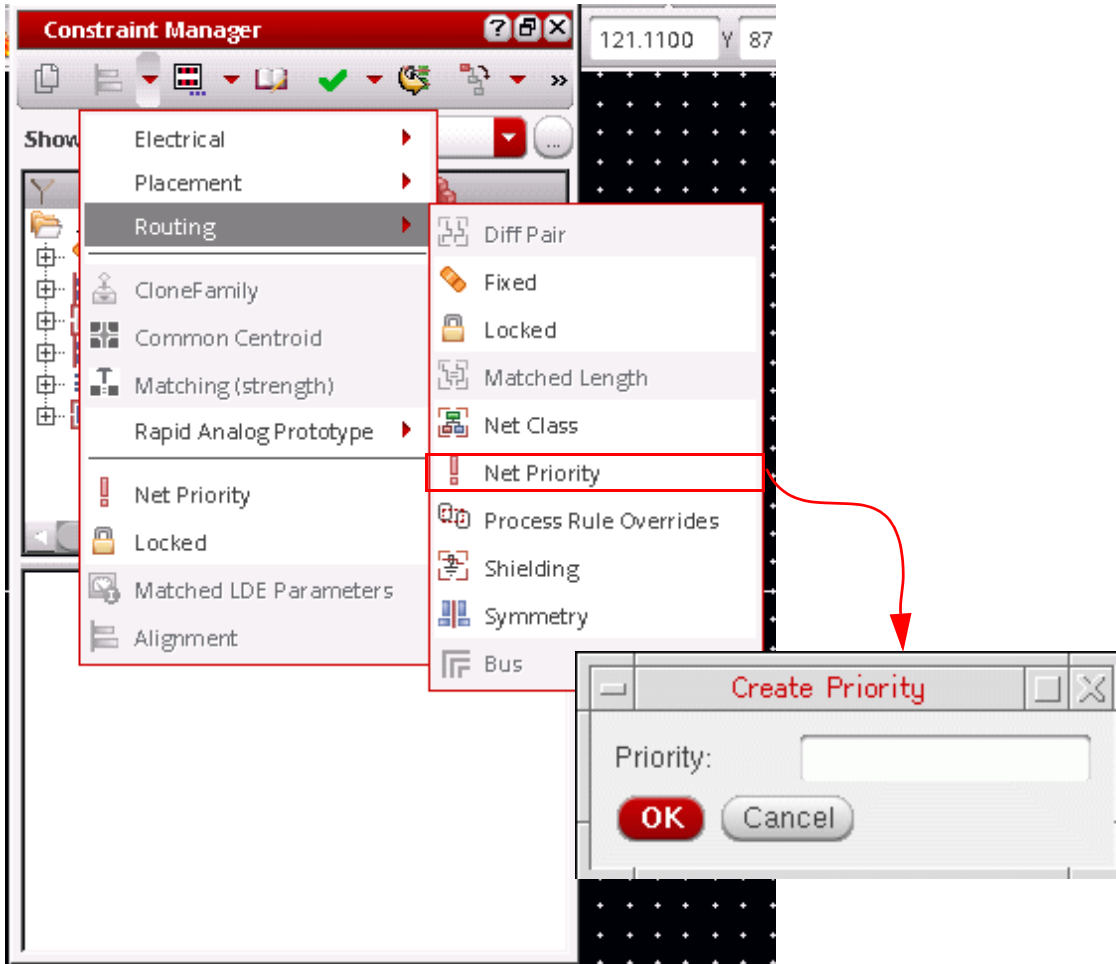


2. In the Navigator assistant, select the first net in your ordered net constraint.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

3. In the Constraint Manager assistant, pull down the constraint list and choose *Routing – Net Priority*.



The *Create Priority* dialog is displayed.

4. Type in the priority for the selected net (for example, 1).

Note: Zero is not a valid value for the net priority. Enter a non-zero integer between -128 and 128.

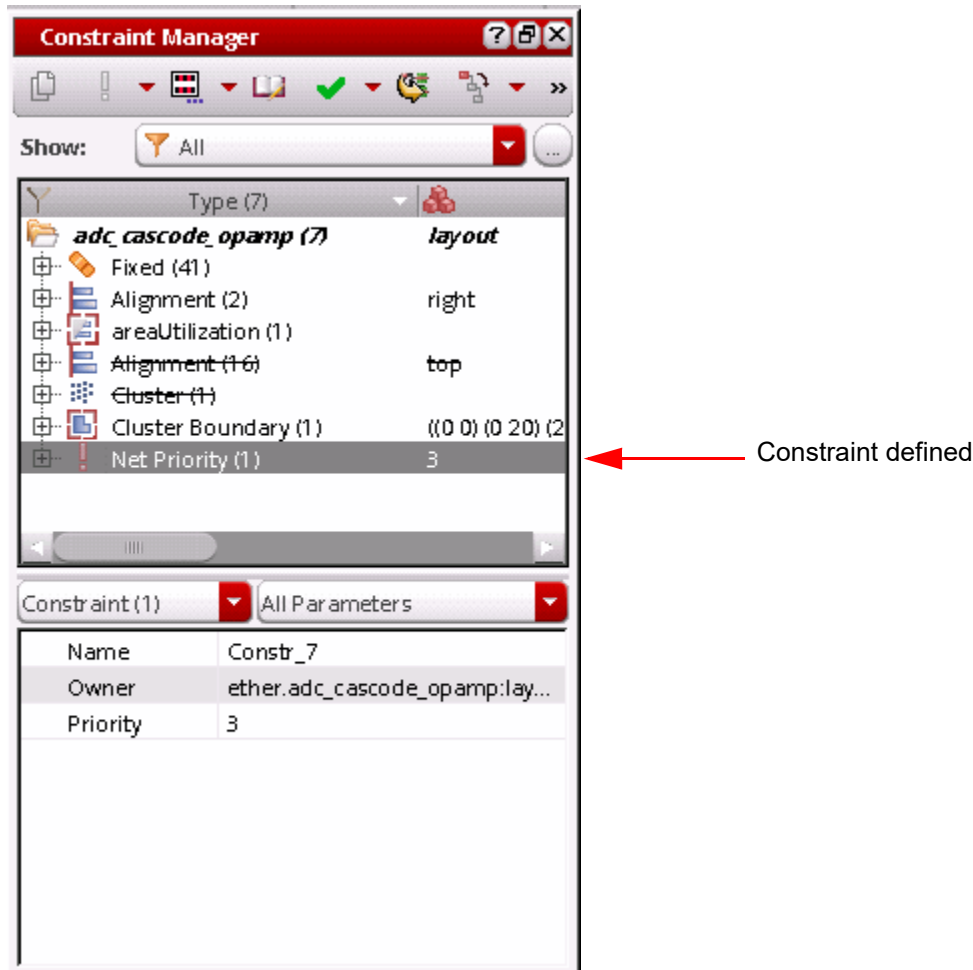
5. Click *OK* to create the constraint.

It is added to the list in the Constraint Manager assistant.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

- Repeat steps 2 thru 5 for each additional net you want to add to the ordered net constraint.



Note: The nets need not belong to the same logical bus. If you assign the same priority to two nets, both nets are added to a single *Net Priority* constraint.

For more information on Constraint Manager functionality, see [The Constraint Manager Assistant](#) in the *Virtuoso Unified Custom Constraints User Guide*.

Creating a Net Class Constraint

A *Net Class* constraint defines a group of nets that have a specific set of common constraints (typically process rule overrides) on or between its members, or with respect to other nets or groups of nets in the design. *Net Class* constraints are honored by the Virtuoso Floorplanner.

To create a *Net Class* constraint:

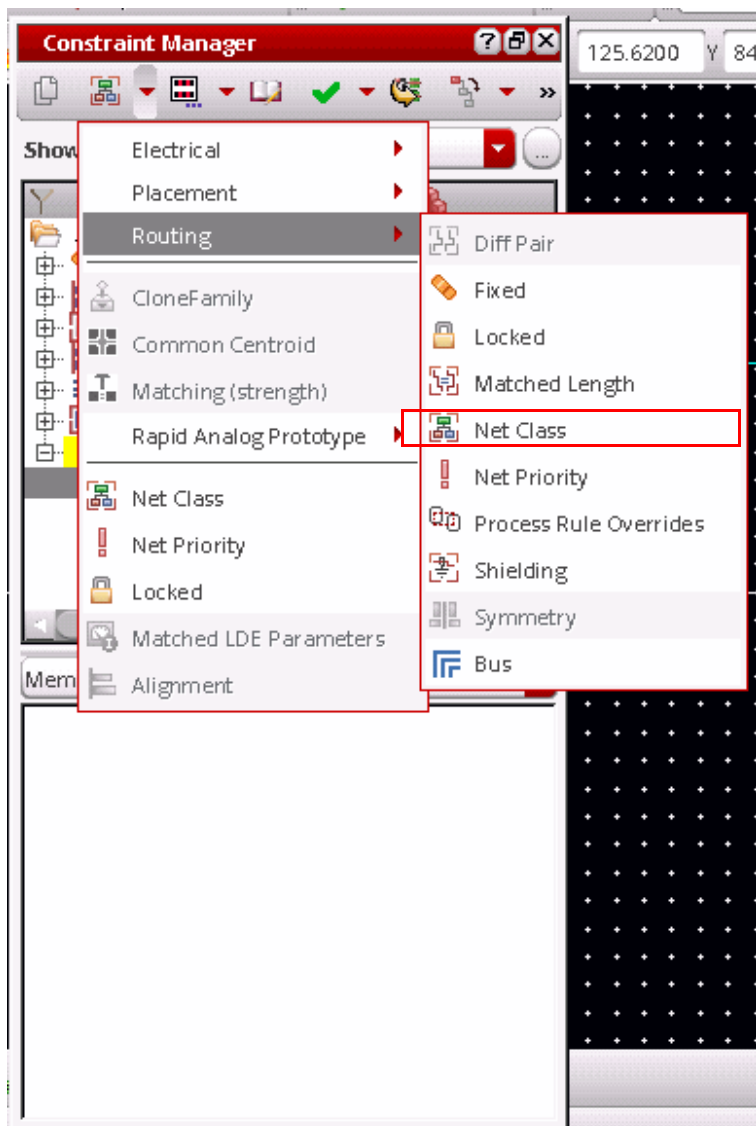
Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

1. In the Navigator assistant, select the nets you want to group into a *Net Class* constraint.
2. From the layout window menu bar, choose *Windows – Assistants – Constraint Manager*.

The Constraint Manager assistant is displayed.

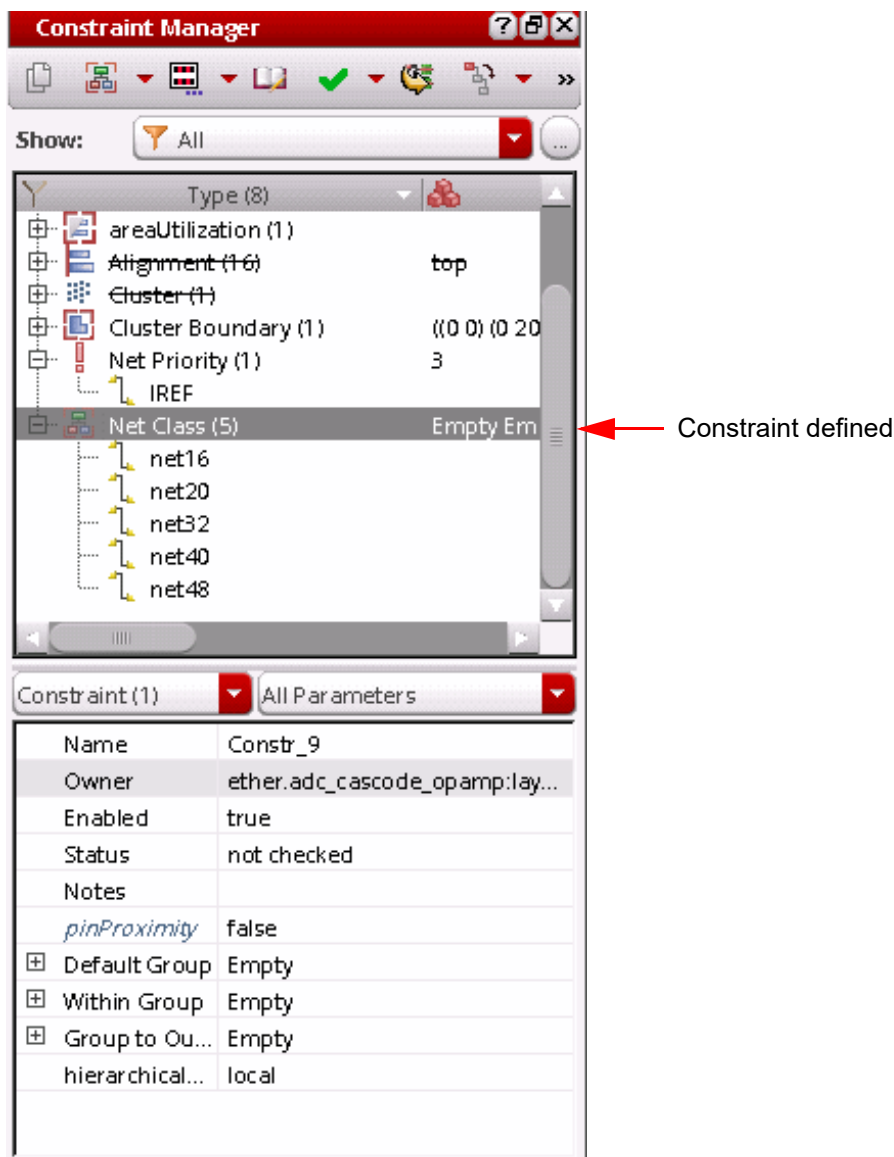
3. In the Constraint Manager assistant, pull down the constraint list and choose *Routing – Net Class*.



The *Net Class* constraint is created with the members you selected, and the constraint is added to the list in the Constraint Manager assistant.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant



For more information on Constraint Manager functionality, see [The Constraint Manager Assistant](#) in the *Virtuoso Unified Custom Constraints User Guide*.

Making Net Connections

There are two main ways to create interconnect in Layout XL.

- Create a wire using the *Create – Wire* command.

For more information, see the [Interactive Wire Editing](#) in the *Virtuoso Space-based Router User Guide*.

Virtuoso Unified Custom Constraints User Guide

The Constraint Manager Assistant

- Create objects such as paths, rectangles and polygons using the *Create Path*, *Create Polygon*, and *Create Rectangle* commands.

For more information, see [Creating Objects](#) in the *Virtuoso Layout Suite XL: Basic Editing User Guide*.

The Circuit Prospector Assistant

An Introduction to the Circuit Prospector Assistant

Note: The *Circuit Prospector* recognizes structures using place-holder names for device, parameter, terminal and net names. As each Process Design Kit (PDK) and design library can use any names, each PDK and library must register the names used by each object for the respective place-holder names. For more information see [*PDK Setup for the Circuit Prospector*](#) in the *Virtuoso Unified Custom Constraints Configuration Guide*.

The *Circuit Prospector* assistant can be used to detect circuit structures, devices and nets based on certain criteria, and optionally apply a default set of constraints to them. It can also be used to set properties, or to visualize various groups of objects; acting as an additional *search* facility. Moreover, it can also be used for debugging, and to also apply specific actions to selected results.

Note: For more overview information see also [Circuit Prospector Overview](#).

In general, designers will want to apply a set of constraints to common circuit structures, related devices (such as same size, area, or type), devices with particular characteristics/ connectivity, or symmetrical devices/nets/pins with particular properties.

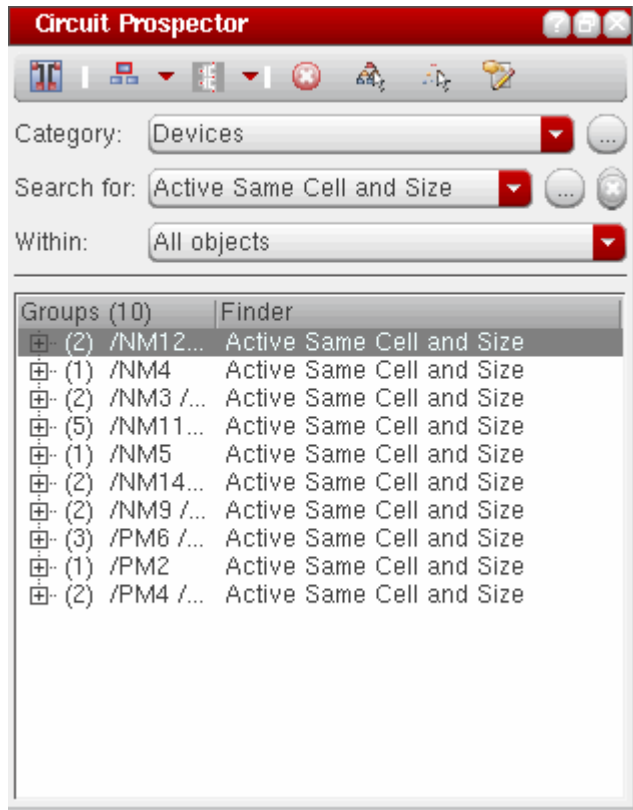
However, performing this task manually and repetitively for multiple designs can be time consuming and error prone. As an aid to this task, the *Circuit Prospector* Assistant provides a flexible framework for capturing the information that a designer is looking for, locating structures and devices, and applying appropriate constraints to them, and consequently greatly accelerating the constraints entry process.

The *Circuit Prospector* provides a set of common structure finders that can be modified by the users to suit their design needs. Users can also create their own finders or capture any structure in the design and save it as a custom finder. The finders are grouped together in 'Categories' that can be customized as well. (See [Creating and Editing Categories](#) and [Creating Finders](#)).

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

Iterators are SKILL functions that analyze cellview devices, nets, and pins in a variety of ways, for example, you can iterate all selected objects, instances, or symmetrical devices.



Note: As with [The Constraint Manager Assistant](#) there can only be one *Circuit Prospector* assistant visible in each session window.

Important

For information on SKILL commands that can be used for *Circuit Prospector* assistant customization, see the [Custom Constraints Functions](#) chapter in the [Virtuoso Unified Custom Constraints SKILL Reference](#).

Motivation

The following are some of the reasons why the *Circuit Prospector* is an essential tool for facilitating quick constraint entry:

- Most design tools are constraint driven.
- Analog circuits require a large number of constraints on devices and nets. The number of constraints is further on the rise with increasing process and design complexity.

- Entering constraints manually is time consuming and error prone.

The *Circuit Prospector* can help accelerate the constraint entry process significantly by retrieving and automatically grouping all design objects by common criteria, visualizing them, and suggesting relevant and corresponding constraints or properties.

Storage

All user-defined or modified Finders, Structures, Iterators and Constraint Generators are stored in your local `.cadence/dfII/ci` directory. The Cadence File Search mechanism is used to locate all `.cadence` directories and all finders, structures, iterators, and generators are loaded as they are found.

The files in your local `.cadence` directory are, by default, the last to be loaded and can override any finders, structures, iterators and generators that have been defined in other `.cadence` directories.

Accessing the Circuit Prospector

You can access the dockable *Circuit Prospector* assistant by either:

- Selecting *Window – Assistants – Circuit Prospector*.
- Selecting *Window – Workspaces – Constraint-Helper*.
- Selecting *Circuit Prospector* from the menu bar/toolbars context-sensitive menu.
- Selecting the *Constraint-Helper* workspace from the *Workspace Configuration* pull-down menu on the toolbar.

For more information see the [Constraints-Helper Workspace Configuration](#).

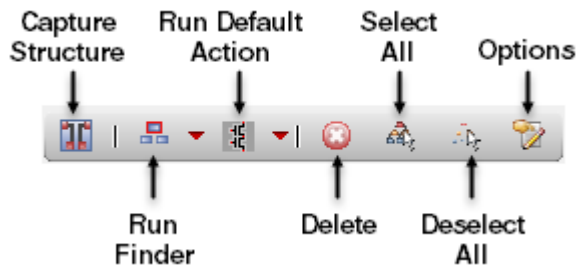
Circuit Prospector User Interface

The *Circuit Prospector* Assistant has three main components:

- Circuit Prospector Toolbar
- Set of Drop-Down List Boxes
- Results Table

Circuit Prospector Toolbar

The first component of the Circuit Prospector is the toolbar at the top, which contains the following options:



Capture Structure

Capture Structure based on the settings and selections made in the *Circuit Prospector*.

Selecting this option will display the Capture Circuit Structure Form.

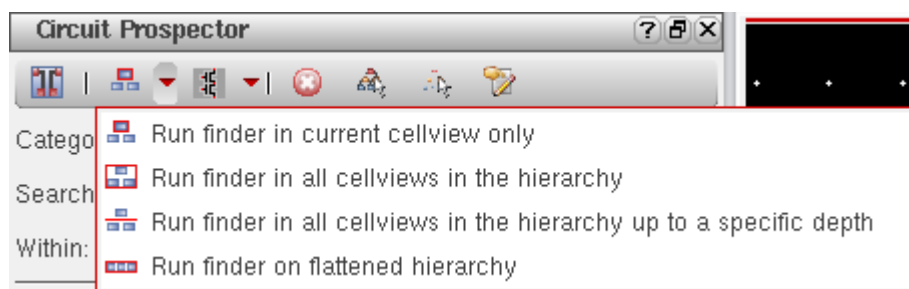
For more information, see Step 4: Apply default action/ constraints to the selected objects.

Run Finder

Run finder... based on the finder selection in the *Search for* drop-down list box.

Note: The *Run finder...* and *Run Default (Constraint Generation) Action* drop-down menu on the *Circuit Prospector* toolbar become active only after you select a finder in the *Search for* drop-down list box.

Finders can be run on just the current schematic (the default), or the whole schematic, starting from the current cellview. See also ciRunFinder in the *Virtuoso Unified Custom Constraints SKILL Reference*. Choose from the following run finder options:

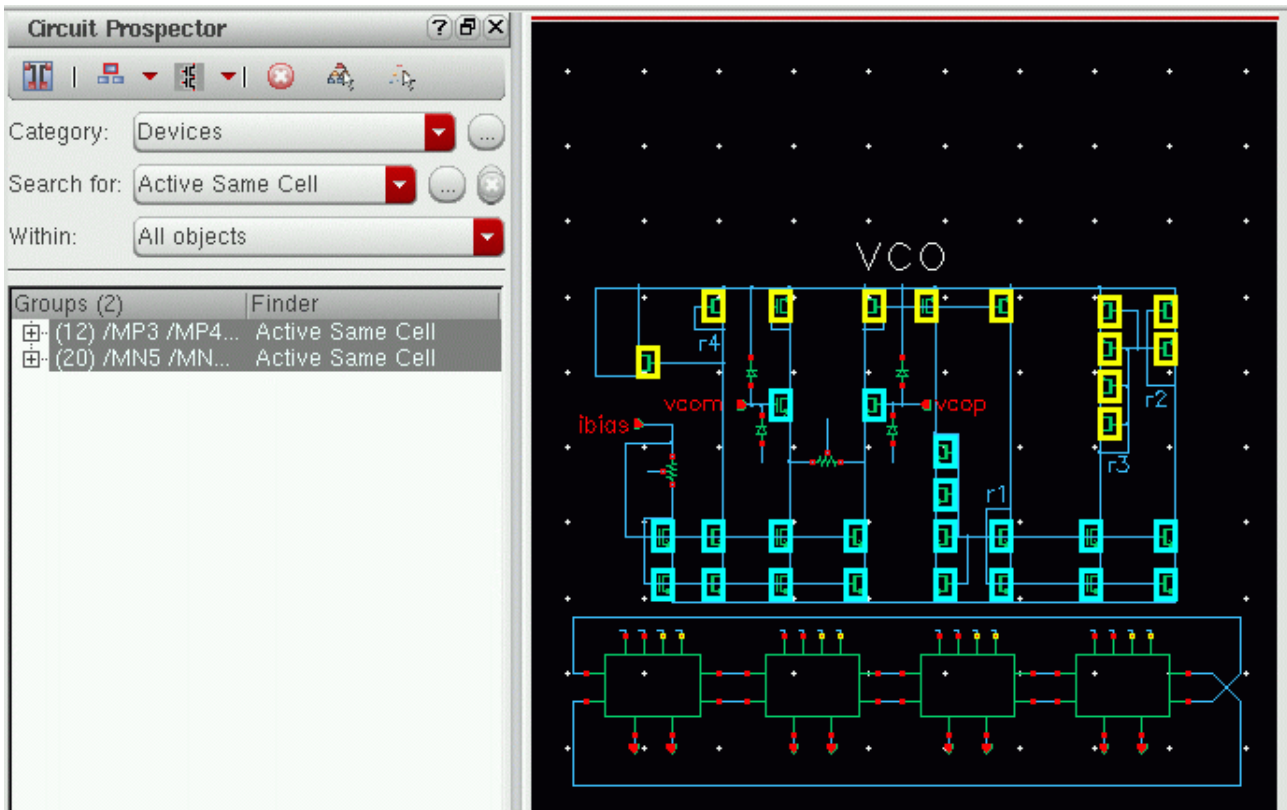


Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

■ *Run finder in current cellview only (the default mode)*

The figure below shows sample results when this run finder option is selected.



■ *Run finder in all cellviews in the hierarchy*

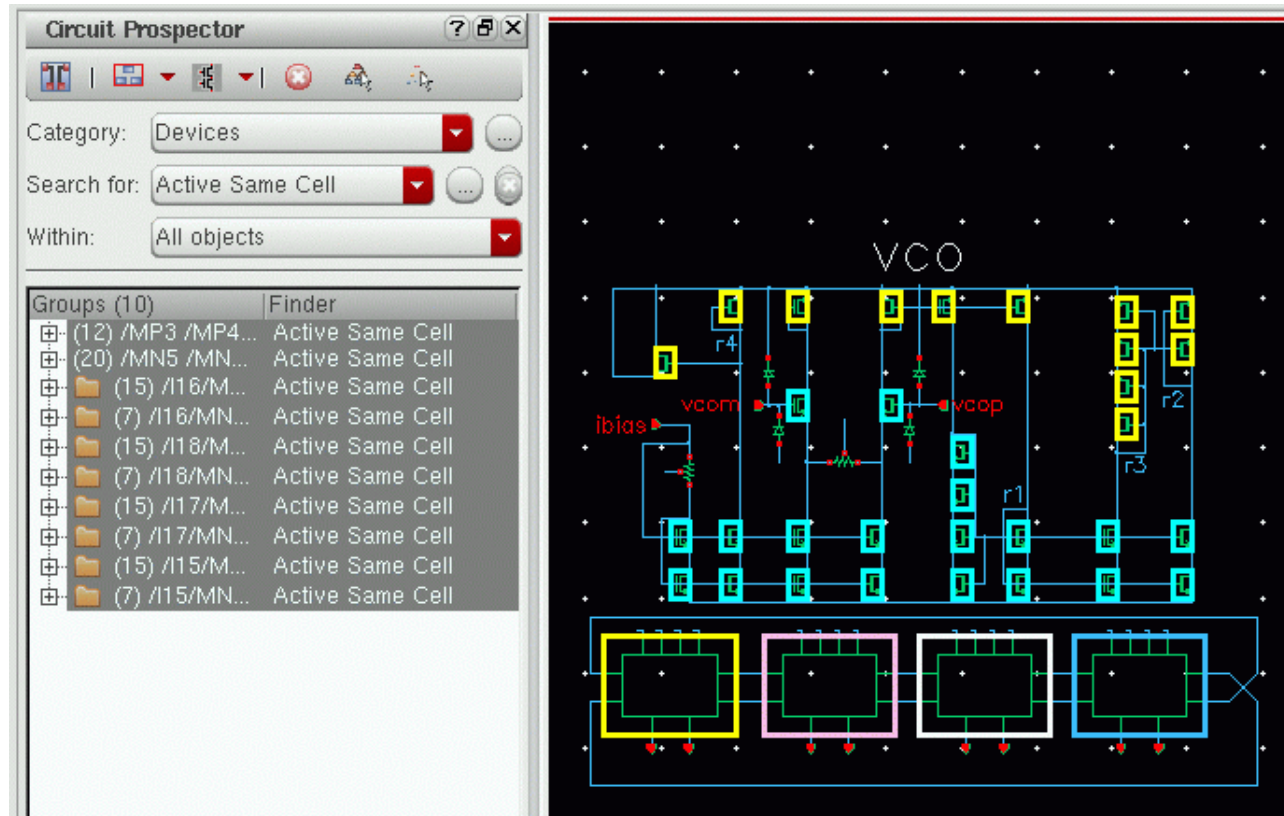
Runs the finder on the current cellview and all other cellviews in the hierarchy that are below the current cellview. The finder results for cellviews that are below the cellview will be grouped into directories based on the hierarchical level in which they were found.

In this mode, the finders are run on each cellview in the hierarchy in isolation. For example, the *Active Same Cell and Size* finder groups all devices that have the same cell and size on a cellview-by-cellview basis.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

The figure below shows sample results when this run finder option is selected.



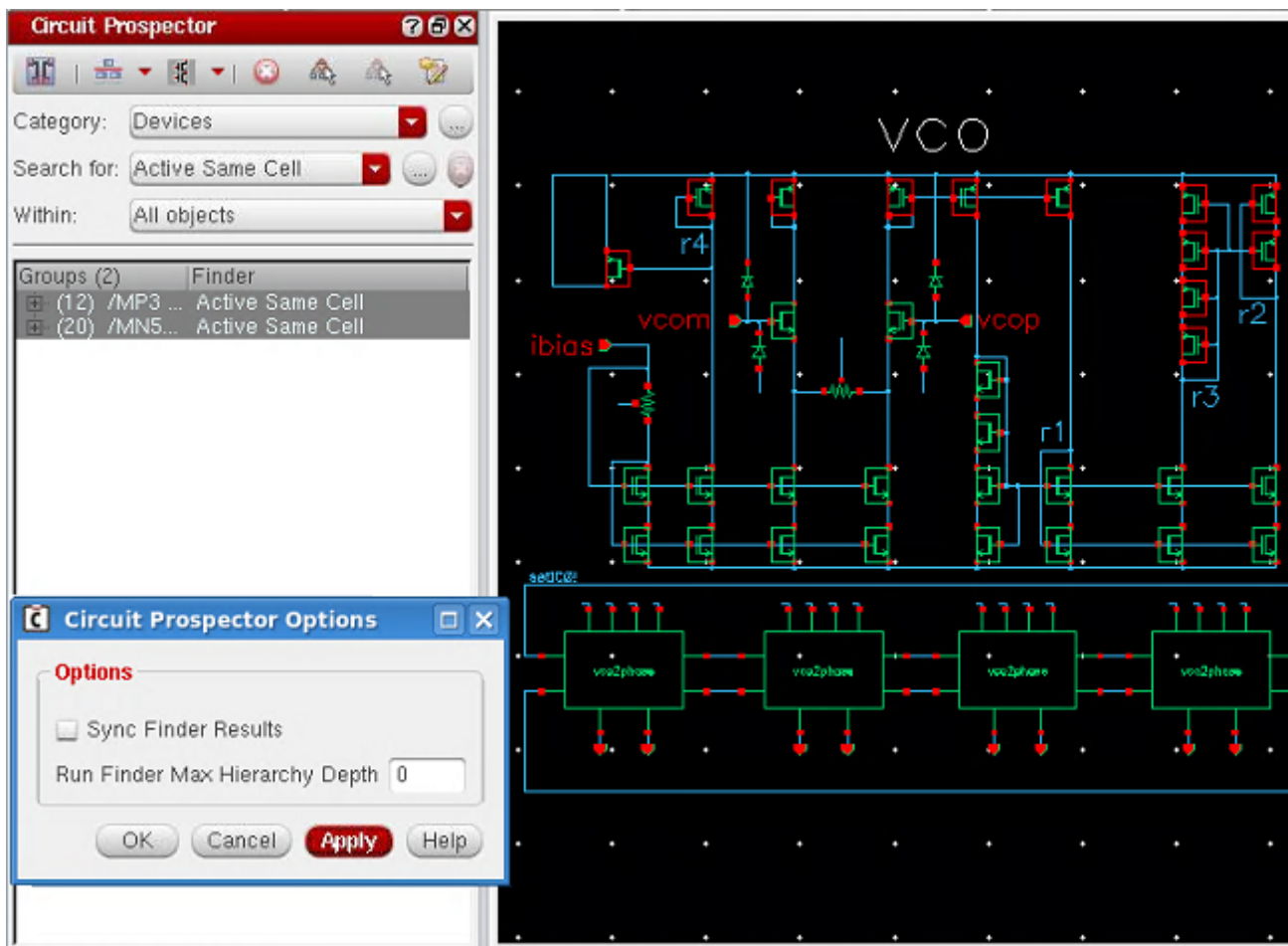
■ *Run finder in all cellviews in the hierarchy up to a specific depth*

Runs the finder on the current cellview and all other cellviews in the hierarchy that are below the current cellview up to the maximum depth specified in the *Run Finder Max Hierarchy Depth* field on the Circuit Prospector Options form.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

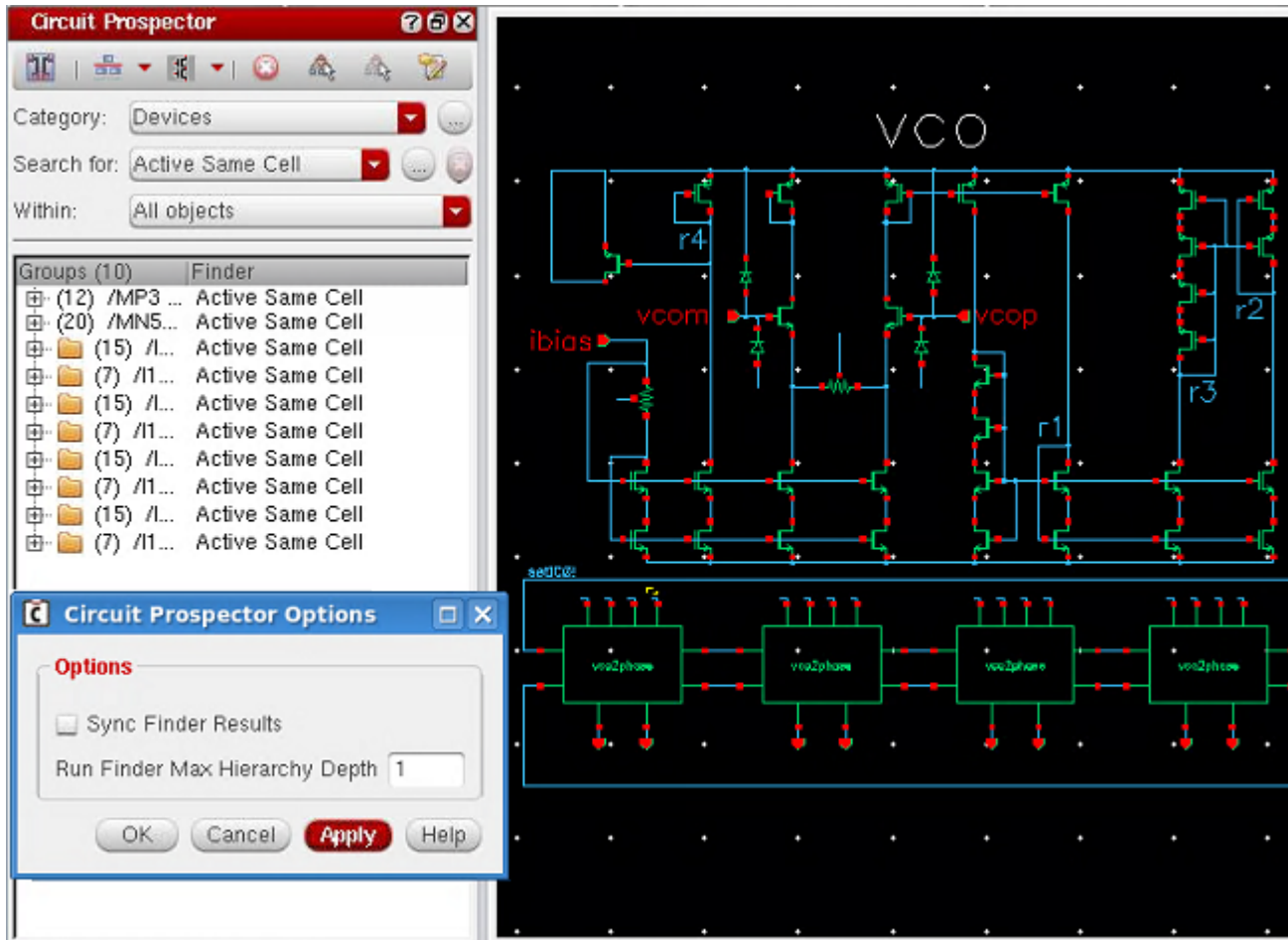
The figure below shows sample results when this run finder option is selected with the maximum hierarchy depth set to 0.



Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

The sample results in the figure below show the difference when the same run finder option is rerun keeping the *Category* and *Search for* selections the same, but with maximum hierarchy depth set to 1 instead of 0.



■ Run finder on flattened hierarchy

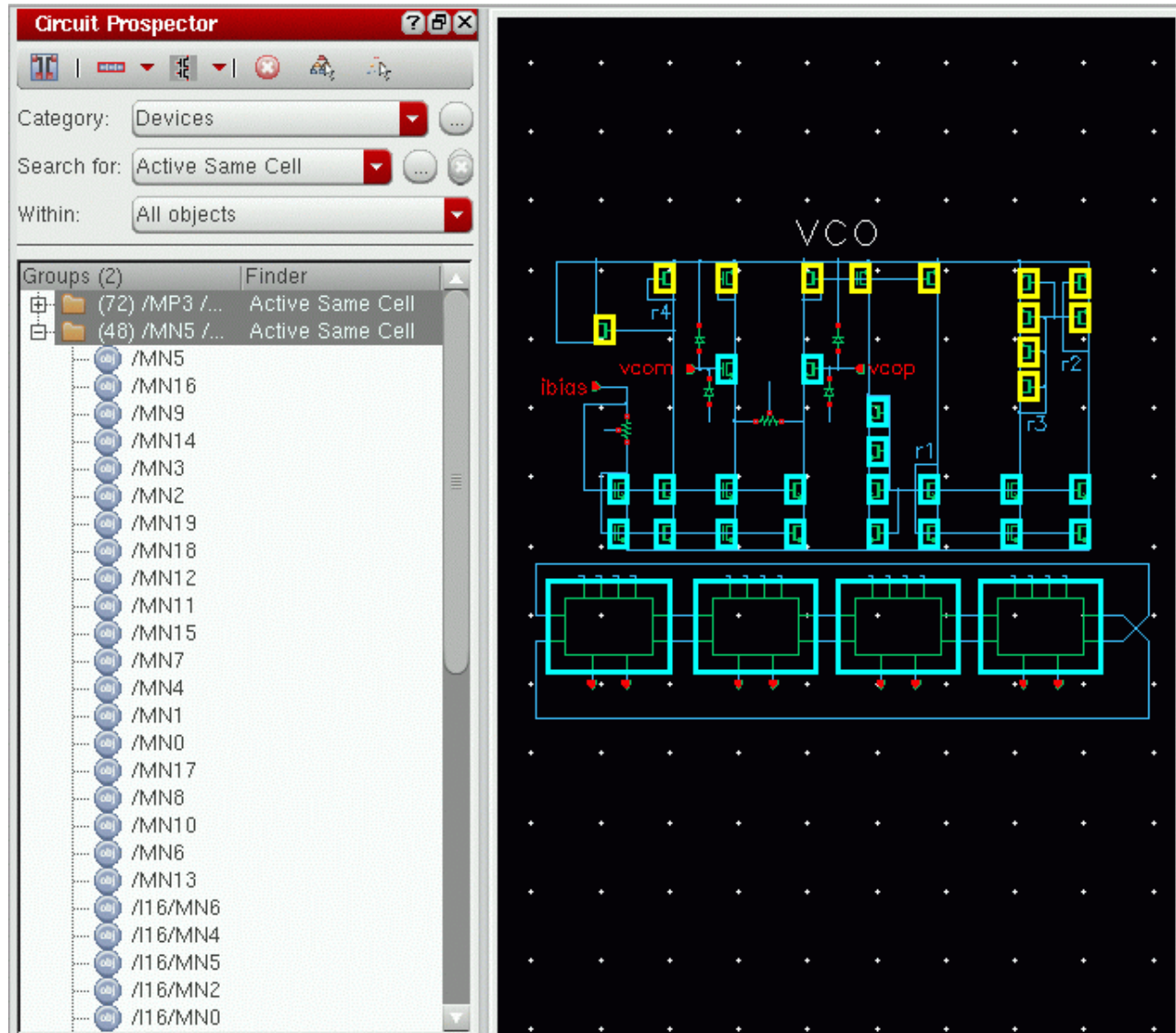
Groups all matching results in the same directory, whatever the instance and hierarchical level at which they were found (according to the opened Hierarchy Editor or CPH configuration, and the corresponding leaf cells).

Finders are run on all cellviews in the hierarchy, but across hierarchical boundaries. For example, the *Active Same Cell and Size* finder will group all the devices that have the same cell and size, regardless of where they are in the hierarchy relative to the current cellview.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

The figure below shows sample results when this run finder option is selected.



Important

The *Run finder on flattened hierarchy* option enables you to select all of the cells of the same type in the current schematic regardless of their hierarchical level or parent (sub)cell. Whereas, the *Run on all cellview in the hierarchy* option creates a group of the same based on the hierarchical levels.

In case any of these options is selected, the Circuit Prospector conserves the results if descend edit done in the same tab or a new tab. However, it does not conserve the results in a new window.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

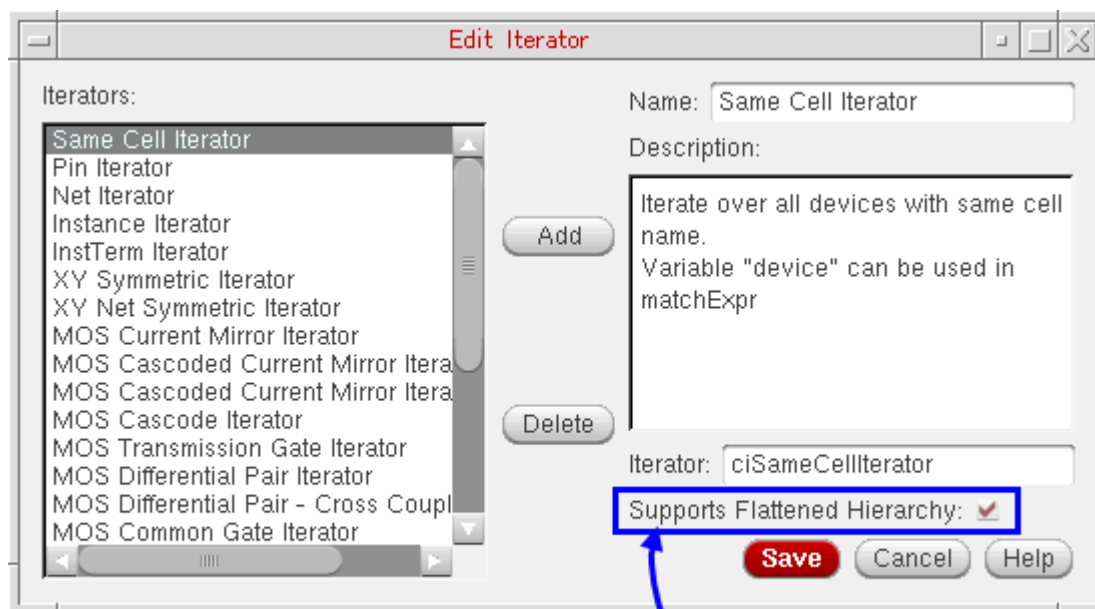
When running hierarchically, the current configuration is used. If there is no configuration available, the switch list is used. Additionally, for larger designs, that the *Circuit Prospector* might take more than a few seconds to run on, a progress bar is displayed to indicate the current progress. If you want to interrupt this process, select **Ctrl+C**.



Tip

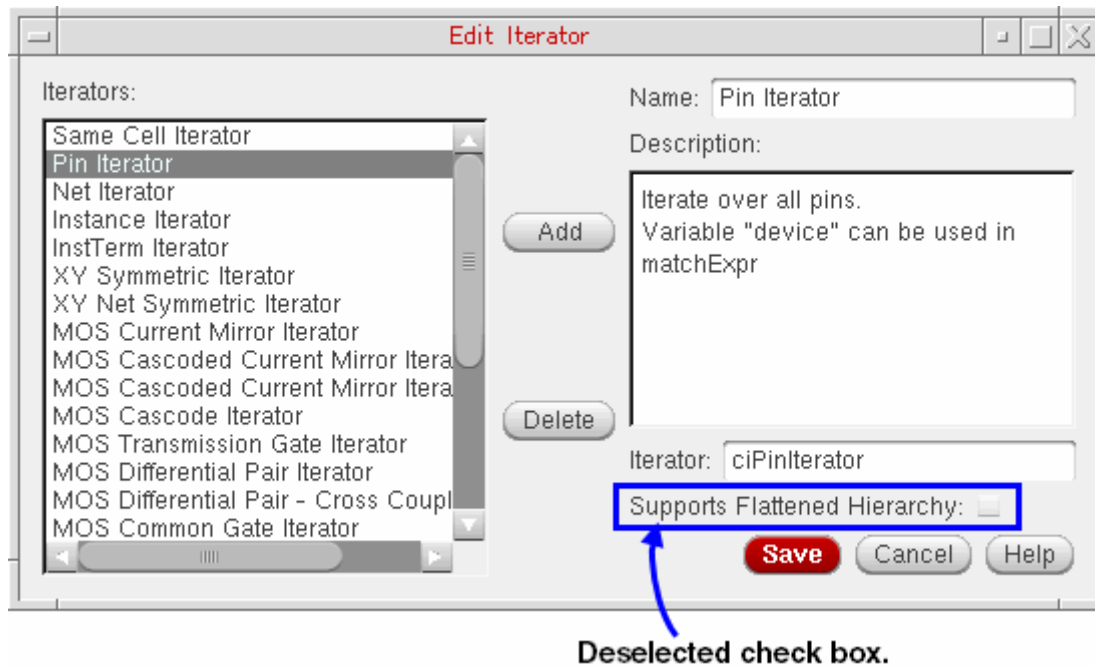
All finders can be run in all cellviews in the hierarchy, but only a few finders can be run on a flattened hierarchy. To run on a flattened hierarchy, the iterator must be defined accordingly. If an iterator supports a run on a flattened hierarchy, the respective option has to be set in the Edit Iterator form. If an iterator does not support a run in a flattened hierarchy, this option will be grayed out.

Figure 2-1 Running Finder on Flattened Hierarchy Supports Iterators with Flattened Hierarchy



Selected check box.

Figure 2-2 Flattened Hierarchy Not Supported



Run Default (Constraint Generation) Action

The options available in this menu (default constraint generation) are dependent upon the finder selected in the *Search* for drop-down list box and the *Groups* selected in the *Circuit Prospector* results table.

Note: A finder must be selected and there must be one resultant group for this option to be enabled.

The first choice available in the menu (for example *Generate Cluster Constraint for the Current Cellview*) runs the default action and creates constraints only in the current cellview. If the finder was run hierarchically, constraints will have hierarchical path members, for example, I16/I22/MN1 and I16/I22/MN2.

The second choice in the menu runs the default action in the cellview at the hierarchical level in which the finder found the result. In this case, constraints would be created in the cellview for I16/I22, and will have members MN1 and MN2.

Note: The corresponding cellviews must be open in write mode.

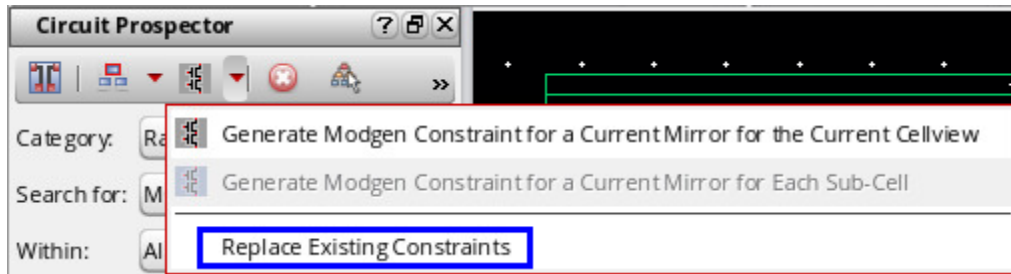
Currently selected finder identifies the type of constraints that are created and subsequently added to the *Constraint Manager* or the properties that are generated after selecting *Run*

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

Default Action. This is because the default action is specified as part of the finder definition (see also [Creating Finders](#)).

The last option in this menu is *Replace Existing Constraints*, as shown below.



Clicking this option selects it as your choice for the consecutive constraint generator runs. Subsequently, running a constraint generator for the current cellview or for each sub-cell, first removes all existing constraints and then, creates the new constraints.

Delete

Use this menu to delete result entries from the results table.

Select All

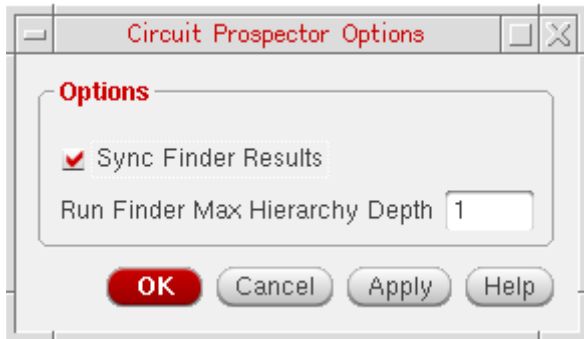
Use this menu to select all result entries in the results table.

Deselect All

Use this menu to deselect all result entries from the results table.

Options

Clicking this menu item displays the following Circuit Prospector Options form:



Use this form to set the following:

- ***Sync Finder Results:*** Select this check box to synchronize the results listed in the Circuit Prospector based on the selected category, finder, and filter criteria with the Constraint Manager and the schematic layout canvas.
- ***Run Finder Max Hierarchy Depth:*** Enter an integer value to specify the maximum depth up to which the finder should run in a hierarchy.

Set of Drop-Down List Boxes

The second component of the Circuit Prospector, right under the toolbar, are a set of drop-down list boxes to select categories of constraints, apply finders, and choose those objects to be constrained. The drop-down list boxes are:

- ❑ ***Category*** (for more information see [Step 1: Select a circuit prospector category and Categories](#))
 - ❑ ***Search for*** (for more information see [Step 2: Select a finder to search for relevant design objects and Finders](#)).
- Note:** This section also contains button options to edit or delete finders.
- ❑ ***Within*** (for more information see [Step 3: \(Optionally\) Filter/ select the found objects to apply constraints to](#)).

Results Table

The third component is a results table that lists, in tree display format, all the constrained groups and the finder that they belong to.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

The results table contains the following columns:

- The *Groups* column which displays the number of constrained groups and lists the current objects that will be included when you choose to *Capture Structure*.

Note: You can click the “+” or “-” icons to expand or collapse the group contents.

- The *Finder* column which details the particular finder a group belongs to (see [Finders](#)).

Important

Selecting a result in the results table of the *Circuit Prospector* will automatically pan to the object in question in the design canvas.

The Circuit Prospector Constraint Creation Flow

The *Circuit Prospector* is used to repeatedly search for devices, pins, and nets to constrain based on specific, customizable criteria. The search results can then be browsed and filtered, before constraints are applied.

The *Circuit Prospector* constraint creation flow can be broken down into 4 main stages:

- Step 1: Select a circuit prospector category
- Step 2: Select a finder to search for relevant design objects
- Step 3: (Optionally) Filter/ select the found objects to apply constraints to
- Step 4: Apply default action/ constraints to the selected objects

Important

Steps 1-4 can be repeated with the same or different finders until all constraints are set or all expected actions are run.

Step 1: Select a circuit prospector category



The first thing a designer should do, as part of the *Circuit Prospector* constraint creation flow, is to select a *Category* from the list of available assistants (see [Categories](#)).

Each *Category* contains a collection of related [Finders](#) (search algorithms).

Once a *Category* has been selected, a list of filtered finders is displayed in the *Search for* drop-down list box.

A *Category* comprises of a collection of related [Finders](#) that are search routines to look for devices, pins, or nets that meet certain criteria, for example, common circuit structures, related devices (same size, area, type, or shared connectivity), devices with certain

characteristics or shared connectivity, symmetrical devices with particular properties and so on.

You should now continue to [Step 2: Select a finder to search for relevant design objects](#)

Step 2: Select a finder to search for relevant design objects

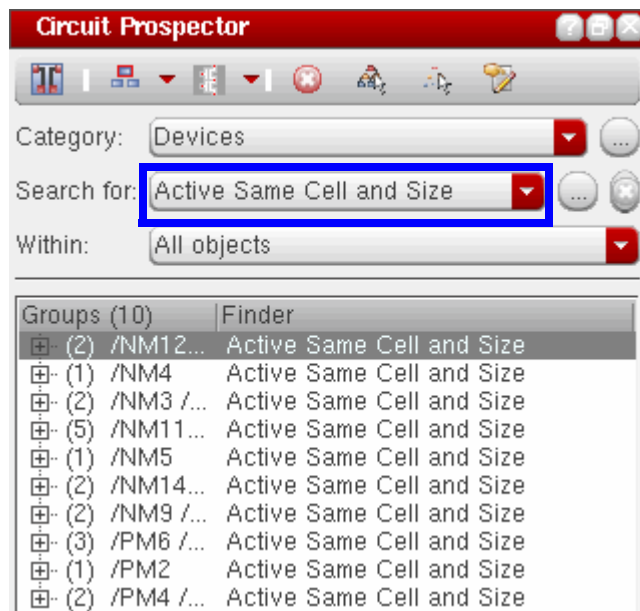


Figure 2-3 Active Same Cell Name and Size finder selected and current results shown

After selecting a *Category*, you must now choose from a range of related *Finders* (search algorithms) from the *Search for* drop-down list box. If ALL is selected from the list, all the finders under the current category will be run sequentially in the order of their listing in the *Search for* drop-down list box.

Important

For information on running finders in the current cellview only, in all hierarchy cellviews, or on a flattened hierarchy, see [Run Finder](#).

As soon as the finder is selected from the *Search for* drop-down list box, the results (objects or groups of objects that meet the search criteria) are displayed in the results table in the same order. It should be noted that by default the finder runs only in the current cellview. To run the finder hierarchically or in a flattened hierarchy mode, follow the steps outlined in the previous section.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

Note: The finder options listed are dependent upon the selected *Category*.

Clicking the *Edit Finder* browse button will display the Edit Finder form, from where you can amend various attributes of the currently selected finder. For more information see [Edit Finder Form](#).

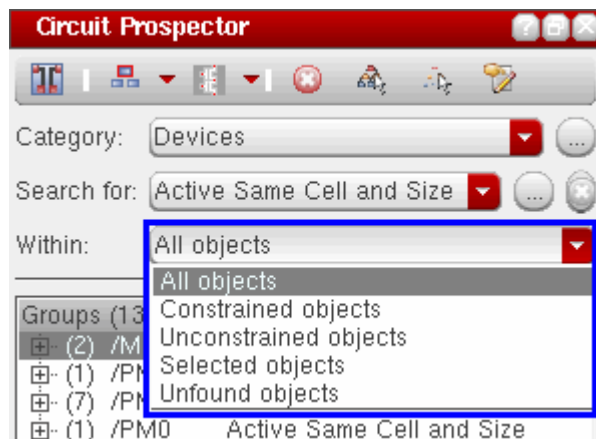
At this point you are now ready to move to [Step 3: \(Optionally\) Filter/ select the found objects to apply constraints to](#).

Step 3: (Optionally) Filter/ select the found objects to apply constraints to

There are two steps to complete when determining what objects a constraint should be applied to:

1. Select Objects to Constrain

The *Within* drop-down list box allows you to filter the objects to which you want to apply the selected finder (search). You have the following filter options:

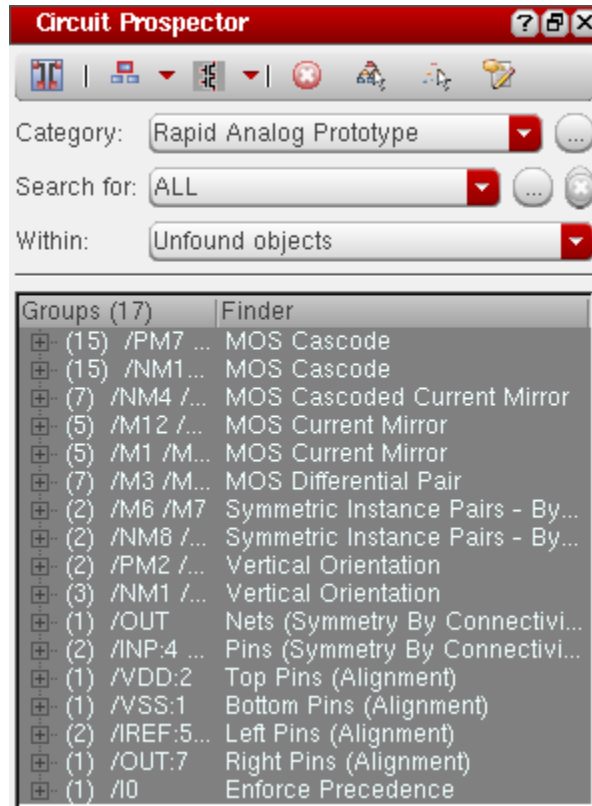


Filter Option	Purpose
<i>All objects</i>	Lists all objects that matched the specified category and finder criteria. The list is displayed irrespective of the fact that the object has any constraint applied or not.
<i>Constrained objects</i>	Lists only those objects that have associated constraints.
<i>Unconstrained objects</i>	Lists only those objects that do not have any associated constraint.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

<i>Selected objects</i>	Lists only the selected objects.
<i>Unfound objects</i>	Lists only unique objects when multiple finders are run sequentially, that is, when you select <i>ALL</i> in the <i>Search for</i> list.



Consider the *Rapid Analog Prototype* category. The *Search for* list for this category lists the finders in the following order: *MOS Cascode*, *MOS Cascoded Current Mirror*, *MOS Current Mirror*, and so on. When you select *ALL* in the *Search for* list and the *Unfound objects* filter in the *Within* list, an object listed by the first finder will not be repeated for the subsequent finders.

However, there is a possibility that when you run the constraint generators for the current cellview, this object might not be used by the corresponding constraint generator. For example, a net object might not be used if *Add Guard Ring* option is not selected in the constraint generator form. Consequently, the object remains unused because no other finder listed it.

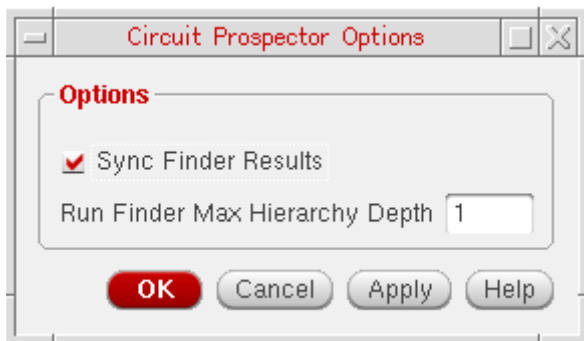
Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

Had the *Unfound objects* filter not been set, this object could have been listed for some other finder and could have been used to generate a constraint of a different type. So, it is noteworthy that the result of using the *Unfound objects* filter might not be a desired trade-off when you might prefer to generate a constraint of some other type on that object.

Normally, the results listed in the Circuit Prospector based on the selected category, finder, and filter criteria are not synchronized with the Constraint Manager and the schematic layout canvas. However, you can enable this type of synchronization by selecting the *Sync Finder Results* check box on the Circuit Prospector Options form that is displayed by one of the following ways:

- Click  (the *Options* icon) on the Circuit Prospector toolbar.
- Right-click the Circuit Prospector assistant and click  Options... from the displayed menu.



As a result, when the *Constrained Objects* or *Unconstrained Objects* filter is selected, the Circuit Prospector finder reruns each time a constraint is created or deleted (manually or using the constraint generator).

Note: By default, the *Sync Finder Results* check box is deselected.

2. Select Groups to Constraint

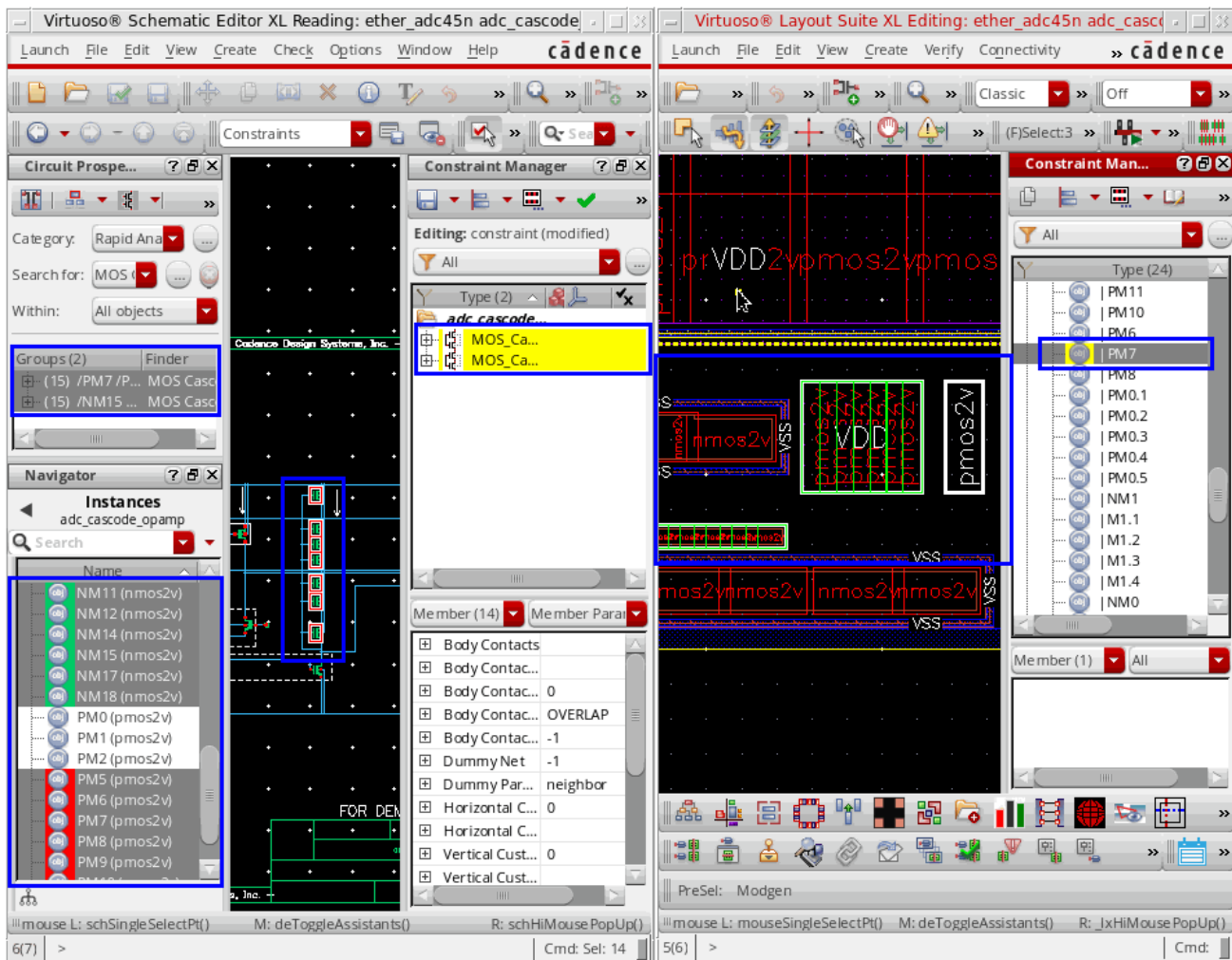
The results area in the lower section of the *Circuit Prospector* groups together each device, net, or pin that meet the search, matched expression and selection criteria. Here, you can choose those groups that you want to apply constraints to. By default, all results in the results table are highlighted and selected.

Selecting entries in either the results section of the *Circuit Prospector*, canvas, or the *Navigator* assistant simultaneously cross-highlights those devices, nets, or pins in the

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

design cellview (the canvas) and other (applicable) assistants such as *Constraint Manager* (in schematic and layout views), *Navigator*, or *Property Editor* (see blue rectangles in the figure below). In addition, the selected result will automatically pan the object(s) in the canvas. This visually helps the designers to avoid creating contradicting constraints on the same device. When you select a device to create a constraint, if the *Constraint Manager* shows the respective constraint which already has this device, you can re-evaluate whether you need another constraint on this device or not. This overall improves the need of constraints in the design methodology.



At this point, you may optionally want to delete (filter) those result entries that you do not want to create constraints for. You can do this by selecting the result(s) using **Ctrl+click**, or you can delete the results that you do not want to constrain, by either pressing the **Delete** key or selecting the *Delete* button from the *Circuit Prospector* toolbar.

Once you have chosen the objects to be constrained and filtered (deleted), the results you can now move proceed to Step 4: Apply default action/ constraints to the selected objects.

Step 4: Apply default action/ constraints to the selected objects

The final step is to apply the default set of constraints (or any other default action such as adding relevant properties to the groups) that is associated with the finder that was run in step 2 on the selected results. It should be noted that users can also pick any relevant constraint from the drop-down list in the Constraint Manager toolbar if they want, instead of applying the default set of constraints. Further, the default action itself associated with a finder is customizable.

To do this:

1. Click the Run Default Action drop-down list box in the *Circuit Prospector* toolbar.

From here you will be displayed with a list of possible constraints that can be created in either the current cellview or for each sub-cell, based on the current *Groups* selection in the *Circuit Prospector* results table.

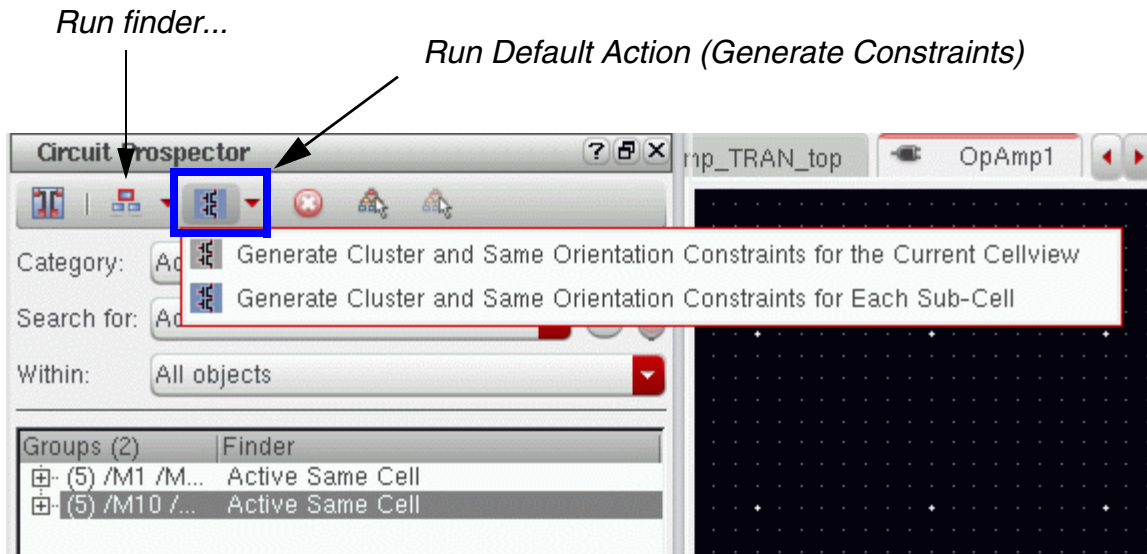
Note: The constraints available for creation in the drop-down list box will depend on the finder selected in the *Circuit Prospector*. This is because the constraint generator is specified as part of the finder definition.

2. Select to create the available constraints (or any other relevant action) for either the current cellview or for each available sub-cell.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

The new constraints will be added to the *Constraint Manager* table. The figure below shows how to choose to create a constraint for the current cellview or each sub-cell.



Note: Each *Generate ... for Each Sub-Cell* option will only be active if *Run finders in all cellviews in the hierarchy* is selected from the *Run finder...* option, as that is the only situation that such an action would be relevant.

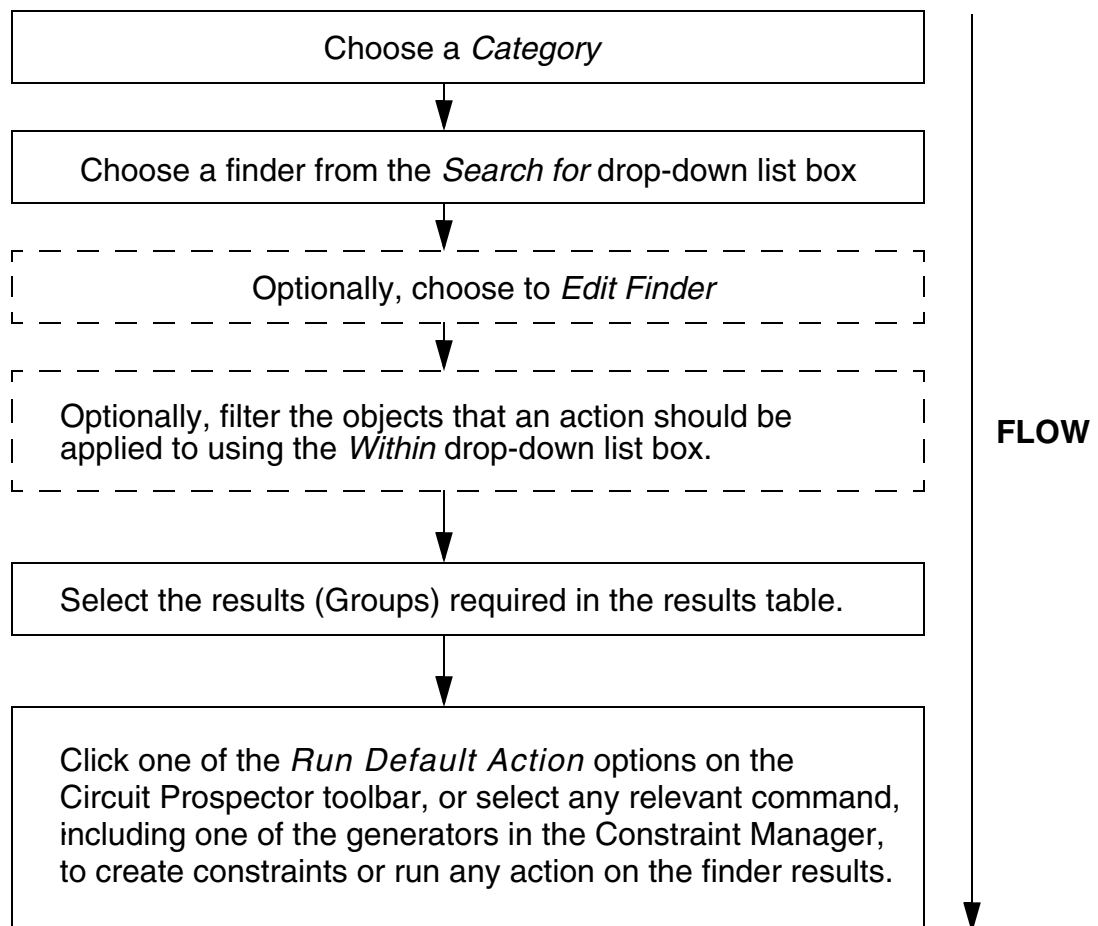
Expanding Constraint Membership Directly on the Canvas

When creating a new constraint, from a selected group in the *Circuit Prospector*, you can expand the membership of the constraint to be created by selecting additional constraint members directly from the design canvas. This therefore allows you to create a new constraint based on a combination of member objects currently selected in the *Circuit Prospector* and also in the design canvas.

Note: A warning message will be issued if you attempt to include canvas member objects, as part of a new constraint, when multiple groups are selected in the *Circuit Prospector*. This is because the *Constraint Manager* does not know which group (that will form a new constraint) that you want the additional canvas object selections to be part of.

Circuit Prospector Flow Summary

The following flow chart summarizes the steps involved in locating objects with the Circuit Prospector, and then generating constraints or running any action on them:



Edit Finder Form

The Edit Finder form is displayed when you select the ... (ellipsis) button to the right of the *Search for* drop-down list box in the Circuit Prospector assistant.

Edit Finder

Name:

Description:

Search Using Iterator:

Match Expression:

Properties: PM1, PM5, PM9, PM11,...

☐	pin#	(various)
☐	instNamePr...	M
☐	Model Name	(various)
☐	Multiplier	(various)
☐	Length	(various)
☐	Total Width	(various)
☐	Finger Width	(various)
☐	Fingers	(various)

Default Generator or Action:

The Edit Finder form is used to edit existing finder or to create new finders. Finders are used to collect objects (instances, nets, or pins) that share certain common characteristics and/or connectivity.

Finders therefore are used to “find” groups of related objects that are required to be constrained. Finders can however also be used for debugging purposes, for example when checking for incorrect or inconsistent property settings or connectivity issues.

Each finder requires an iterator (see [Edit Iterator Form](#)) to iterate (repeatedly examine) the current cellview in a particular way. During this iteration process if the set *Matching*

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

Expression evaluates to a non-nil result, then all objects (pins, nets, or instances) are collected together into groups based on the *Matching Expression* evaluation result.

GUI Item	Description
<i>Name</i>	The name of the currently selected finder.
<i>Description</i>	A description of the currently selected finder.
<i>Search Using Iterator</i>	Specifies which iterator the selected finder should use. Select “...” to invoke <u>Edit Iterator Form</u> to select an iterator to be edited or created.
<i>Matching Expression</i>	An expression to be applied to the iterator to find the required items in the current cellview. This expression can contain references to iterator specific variables. For example, the <i>Same Cell Iterator</i> (see <u>Pre-Defined Iterators</u>) will make available a <i>device</i> variable which will represent the current device in the iteration.
<i>Properties</i>	Displays the current properties for one or more currently selected instances.
<i>Default Constraint Generator or Action</i>	Specifies the constraint generator (or action) that should be associated with the currently selected finder. Once the finder has run you can select the results from the <i>Circuit Prospector</i> results window and then select <u>Generate Constraints</u> from the <i>Circuit Prospector</i> toolbar and the finder’s default constraint generator will generate the constraints. Select “...” to invoke <u>Edit Generator or Action Form</u> .

- For more information see Finders.
- For information on using the Edit Finder form to edit a finder see Editing Finders.

Edit Iterator Form

The Edit Iterator form is displayed when you select the “...” button to the right of the *Search Using Iterator* drop-down list box in the Edit Finder Form.

The Edit Iterator form allows for iterator details to be edited, registered, or deleted. The purpose of this form is to register an iterator function with the *Circuit Prospector*, providing it with a meaningful name and providing a description of what it used for.

Note: Once registered an iterator can be used in one or more *Circuit Prospector* finders.

GUI Item	Description
<i>Iterators</i>	Displays a list of the currently available <u>Iterators</u> .
<i>Name</i>	The name of the currently selected iterator.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

GUI Item	Description
<i>Description</i>	A description of the currently selected iterator. The iterator description should list all variables made available during expression evaluation. For example, the <i>Same Cell Iterator</i> will make a “device” available which represents the current device in iteration. The finder’s matching expression (see Edit Finder Form) can then reference this variable, for example <code>ciIsDevice(device "fet")</code>.
<i>Iterator</i>	The function name of the SKILL iterator function (see the Custom Constraints Functions chapter in the <i>Virtuoso Unified Custom Constraints SKILL Reference</i>). A SKILL iterator function must be defined to take two arguments: <code>cellview</code> and <code>finderExpr</code>. ■ <code>cellview</code> is the current cellview. ■ <code>finderExpr</code> is the finder expression to be evaluated. The function must return a list containing sub-lists of database objects (instance, nets, and pins). The sub-lists represent groups of objects that have the same finder expression evaluation result. For example, where a finder expression of “<code>when(ciIsDevice(device "fet") ciNetOnTerm(device \"G\"))</code>”, the <i>Same Cell Iterator</i> will return groups of fets that share the same net on their gate terminal. For example, <code>list(list(fet1 fet2 fet3) list(fet4 fet5) list(fet6))</code>.
Supports Flattened Hierarchy	Specifies whether the currently selected iterator should support a flattened hierarchy. Note: The <i>Supports Flattened Hierarchy</i> check box is just a read only indicator for both the predefined and user defined iterators.
<i>Add</i>	Adds a new iterator to the <i>Iterators</i> list.
<i>Delete</i>	Deletes the currently selected iterator.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

Edit Generator or Action Form

The Edit Generator or Action form (shown below) is displayed when you select the “...” button to the right of the *Default Generator or Action* section in the [Edit Finder Form](#).

Generators or Actions:

- Matched Parameters
- Matched Size
- Symmetry
- vSymmetry
- hSymmetry
- Alignment
- Orientation
- Module
- Same Cell Name - Group + Relative Orientation**
- Common Gate - Group + Relative Orientation
- Same Well - Group + Relative Orientation
- Module + Matched Parameters
- Group + Relative Orientation + Vertical Alignment
- Group + Relative Orientation + Horizontal Alignment
- Group + Relative Orientation + Matched Parameters
- Symmetry + Matched
- Matched + Distance
- Matched + Symmetry + Distance
- Group
- Supply Route Priority
- Non Supply Route Priority
- Matching (strength)
- Common Centroid
- Inst Symmetry
- Net Symmetry

Name: Same Cell Name - Group + Relative Orientation

Description: Generate Cluster and Same Orientation Constraints

Expression:

```
when( cadr(instsNetsPins) ciConCreateExpanded(cache 'cluster ?members insts ?note strcat("CP: Group together devices with same cell name") ?verbose nil) ) && ciConCreateExpanded(cache 'relativeOrientation ?members insts ?note strcat("CP: Matching orientation for devices with same cell name") ?verbose nil)
```

Buttons: Add, Delete, Save, Cancel, Help

Constraint generators are SKILL expressions that generate one or more constraints.

The Edit Constraint Generator or Action form allows constraint generators to be edited, registered, or deleted from the *Circuit Prospector*. A constraint generator has a *Name*, detailed *Description*, and an *Expression* for generating constraints.

GUI Item	Description
<i>Generators or Actions</i>	Lists all of the available actions and Constraint Generators. Constraint generators are associated with finders as the default constraints for that finder. For example, the <i>Active Same Cell</i> finder has a default constraint generator of “Same Cell Name - Group + Relative Orientation”. Note: See also Pre-Defined Constraint Generators.
<i>Name</i>	Name of the currently selected constraint generator.
<i>Description</i>	Description of the currently selected constraint generator.

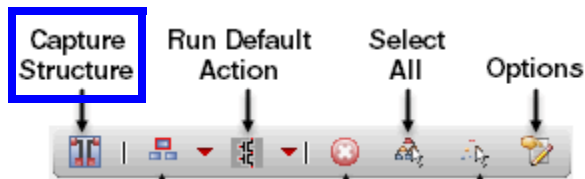
Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

GUI Item	Description
<i>Expression</i>	An expression used to generate constraints. The expression can contain calls to the constraints SKILL function (see the Custom Constraints Functions chapter in the <i>Virtuoso Unified Custom Constraints SKILL Reference</i>) or any other functions that create constraints. The variables “instNetsPins” and “cache” are made available for use when the expression is evaluated. Where: ■ <code>instNetsPins</code> is a list containing three sub-lists for the instances, nets, and pins to be constrained. ■ <code>cache</code> is a reference to the constraints cache. For example, the “Matched Parameters” constraint expression is: <pre>ciConCreateExpanded(cache `matchedparameters ?members append(insts pins ?verbose nil)</pre> The above expression should evaluate to <code>t nil</code> to indicate the success of constraint generation. Note: See also Custom Constraint Generators in the <i>Virtuoso Unified Custom Constraints Configuration Guide</i>.
<i>Add</i>	Adds a new constraint generator to the <i>Constraint Generators</i> list.
<i>Delete</i>	Deletes the currently selected constraint generator.

Capture Circuit Structure Form

You can capture new structures using the Capture Circuit Structure form which is accessible from the *Capture Structure* menu item on the *Circuit Prospector* toolbar.



Clicking the *Capture Structure* menu item displays the Capture Circuit Structure form, as shown below.

The screenshot shows the 'Capture Circuit Structure' dialog box. It has a title bar with the text 'Capture Circuit Structure'. Inside, there are three input fields: 'Name:', 'Type:', and 'Description:'. Below the 'Description:' field is a large text area for an 'Expression (insts, nets, pins) e.g. zerop(length(setof(inst insts inst->orient != car(insts)->orient)))'. At the bottom, there is a table with three columns: 'Objects', 'Match Expression', and 'Repeatable'. The table contains the following data:

Objects	Match Expression	Repeatable
Insts		
+ PM7	@libName == "gpdk045" && @c...	☐
Nets		
Pins		
Constraints		

To the right of the table are four buttons: 'Edit Expression', 'Update From Selected', 'Recapture Constraints', and 'Delete'. At the bottom right of the dialog are four buttons: 'OK' (highlighted in red), 'Cancel', 'Apply', and 'Help'.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

Note: For a list of existing, pre-defined structures, see [Structures](#).

The contents of the Capture Circuit Structure form are described below:

GUI Item	Description
<i>Name</i>	Specify the name of the structure to be created.
<i>Type</i>	Specify the type of structure to be created.
<i>Description</i>	Enter a detailed description of the structure to be created.
<i>Expression (insts, nets...)</i>	Enter an optional and extra match expression that applies to all instances, nets, or pins of the found structure.
<i>Objects/Match Expression/Repeatable table</i>	Displays the structure content and connectivity information, including entries for instances, nets, pins and constraints that are associated with the current structure. Expanding an instance (<i>Insts</i>) in the table displays the instance terminals connected to nets in the structure (terminals that are not part of the structure are not listed). Associated with each instance is also a <i>Match Expression</i> that captures the important properties for that instance within the captured structure. Double-clicking the matched expression for an instance displays the Edit Match Expression Form. Note: The default instance match expression is libName/cellName. Expanding any <i>Constraints</i> item in the table will display all the constraints that exist for the captured nets, instances and pins. If you expand <i>insts</i> or <i>nets</i> in the <i>Object</i> column, check boxes will be displayed in the <i>Repeatable</i> column for each instance or net. Checking any check boxes here will confirm that the particular insts/nets can be repeated <i>N</i> times in the structure (for an example of use see Using Repeatable on a Captured Structure). Note: During pattern matching the basic structure will be grown repeatedly until it can no longer be matched.
Edit Expressions	Invokes the Edit Match Expression form where you can edit a match expression for a captured structure. From this form, you can choose relevant properties to add to the expression.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

GUI Item	Description
Update From Selected	Choose to add further instances, nets, or pins to the structure, by selecting them on the canvas, before selecting the <i>Update From Selected</i> button.
Recapture Constraints	Lets you add further constraints, created in the <i>Constraint Manager</i> assistant, to the captured structure.
<i>Delete</i>	Deletes any constraints, instances, net, or pins that are currently selected in the <i>Object/Match Expression</i> table from the structure.

Capturing New Structures

Note: Before capturing a new structure it is recommended that you enter constraints for the devices of the structure. These constraints will then be captured with the structure and will become the default constraints for that structure.

To capture new structures:

1. Select the instances and/or pins on the device canvas that you want to capture.

Note: Wires do not need to be selected as the connectivity between the selected objects will be captured automatically.

2. Select the *Capture Structure* button on the *Circuit Prospector* assistant toolbar.

This displays Capture Circuit Structure Form.

3. Enter the *Name* of the structure.

4. Enter the *Type* of the structure.

5. Enter a *Description* of the structure.

6. Optionally, edit any matched expressions for current instances (*Insts*).

For more information, see the Edit Match Expression Form and the Edit Terminal Name Expression Form.

7. Optionally, choose to add further instances, nets, or pins to the structure by selecting them on the canvas and then selecting the *Update From Selected* button.

8. Optionally, choose to add further constraints to the captured structure using The Constraint Manager Assistant.

Once these constraints have been created, select the *Recapture Constraints* to update the constraints in the *Objects/Match Expression* table.

9. Click *OK* to capture the structure.

The structure is now saved to the local `.cadence` directory and will be listed in the list of finders for the current category (in the *Search for* drop-down list box).

Note: Before a structure is saved to disk, a connectivity check is performed to ensure that each instance in the structure is connected to at least one other instance in the structure. This is an essential requirement so that the structure recognition algorithm can find the structure. If there are any problems with connectivity, then an error pop-up window is displayed and the connectivity of the structure will need to be changed before it can be saved.



Important

The captured structure device expressions have the `ciMatchedInsts` variable. This variable is a list of the device database IDs that have been matched for the structure so far, for example, to ensure all the devices matched for `t`.

Use the `addParallelDevices` environment variable to control inclusion of parallel devices in the structures during structure matching.

Editing Structures

To edit a structure:

1. Open the schematic from where the structure was captured.
2. Select the structure from the list of finders in the *Search for* drop-down list box.
Capture Circuit Structure Form will be displayed.
3. Optionally, edit the *Name* of the structure.
4. Optionally, edit the *Type* of the structure.
5. Optionally, edit the *Description* of the structure.
6. Optionally, edit any matched expressions for current instances (*Insts*).

For more information see Edit Match Expression Form.

7. Optionally, choose to add further instances, nets, or pins to the structure by selecting them on the canvas and then selecting the *Update From Selected* button.
8. Optionally, choose to add further constraints to the captured structure using The Constraint Manager Assistant.

Once these constraints have been created, select the *Recapture Constraints* to update the constraints in the *Object/Match Expression* table.

9. Click *OK* to save the edited structure.

Deleting Structures

To delete a structure:

1. Select the structure you want to delete from the *Search for* drop-down list box in the *Circuit Prospector* assistant.
2. Click the adjacent *Delete Finder* (cross) button to delete the selected structure.
A pop-up message asking you to confirm deletion is displayed.
3. Select *OK* to confirm structure deletion, or select *Cancel* to cancel structure deletion.
The structure will be removed from the *Search for* drop-down list box.

Using Repeatable on a Captured Structure

Checking any of the check boxes in the *Repeatable* column of Capture Circuit Structure Form will confirm that specific insts/nets can be repeated *N* times in the structure. See the Edit Terminal Name Expression Form for how to specify the way in which repeated device terminals should be connected.

For example:

- If you have captured two transistors *M2* and *M3* (see Capturing New Structures), and their interconnects, you can then make a search more extensive (and for the finder to take the longest list of devices that are connected in the same way (cascode)), by using the *Repeatable* check boxes to instruct which devices and nets can be repeated, and the Edit Terminal Name Expression form to specify how the repeated device terminals should be connected.
- The example below shows that the repeatable sub-circuit is made of *M3*, *gate M3* and *drain M2*.
 - That same sub-circuit is found around *M4*, as the drain of *M3* is connected to the source of *M4* (similar to the *drain* of *M2* in the previous sub-circuit).
 - That same sub-circuit is again found around *M5*, for the same repeatable reason.

Therefore, the whole structure made of *M2*, *M3*, *M4*, and *M5* is recognized as being from the same cascode.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

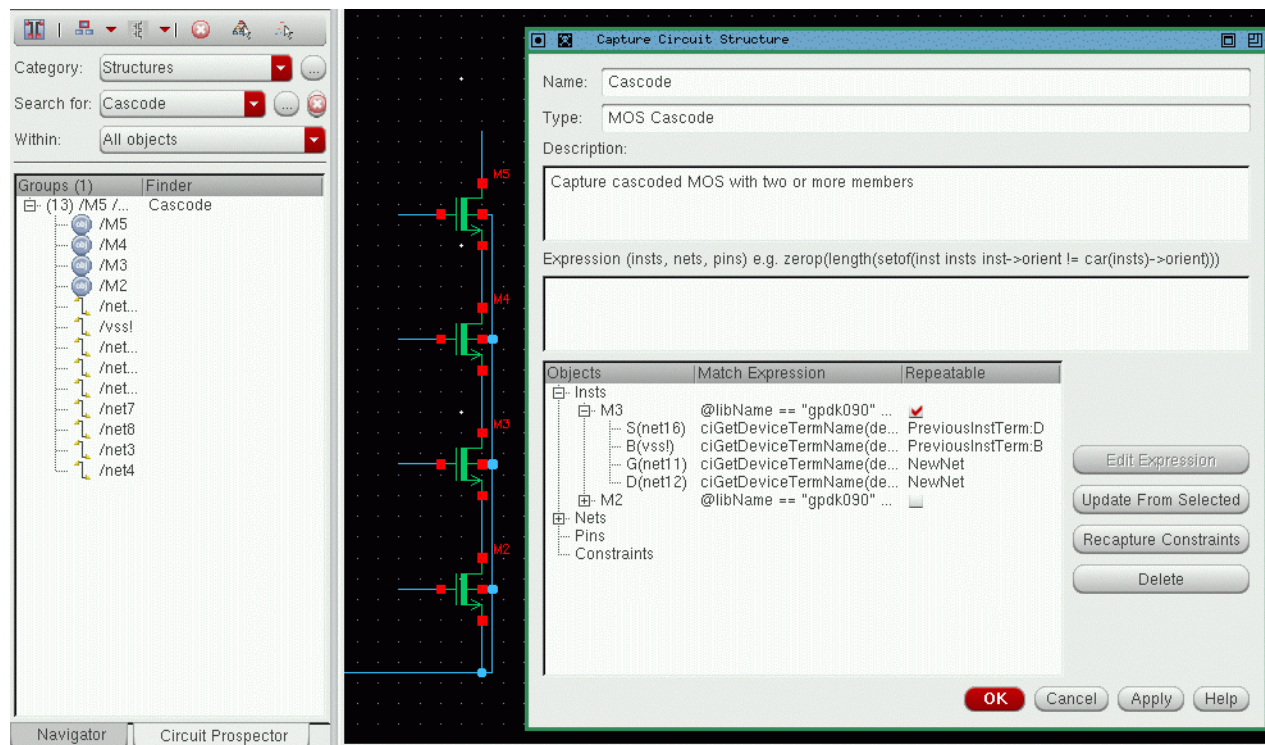


Figure 2-4 Example of Repeatable in Use

Edit Match Expression Form

If you want to edit a match expression for a captured structure, double-click the displayed instance matched expression in the Capture Circuit Structure form or select *Edit Expressions* in the same form. This will display the Edit Match Expression form which allows you to utilize an integrated property editing facility. From here, you can choose relevant properties and add them to the expression.

Note: Matched expressions can also utilize pre-defined helper functions, for example [ciIsDevice](#), to filter the type of objects found.

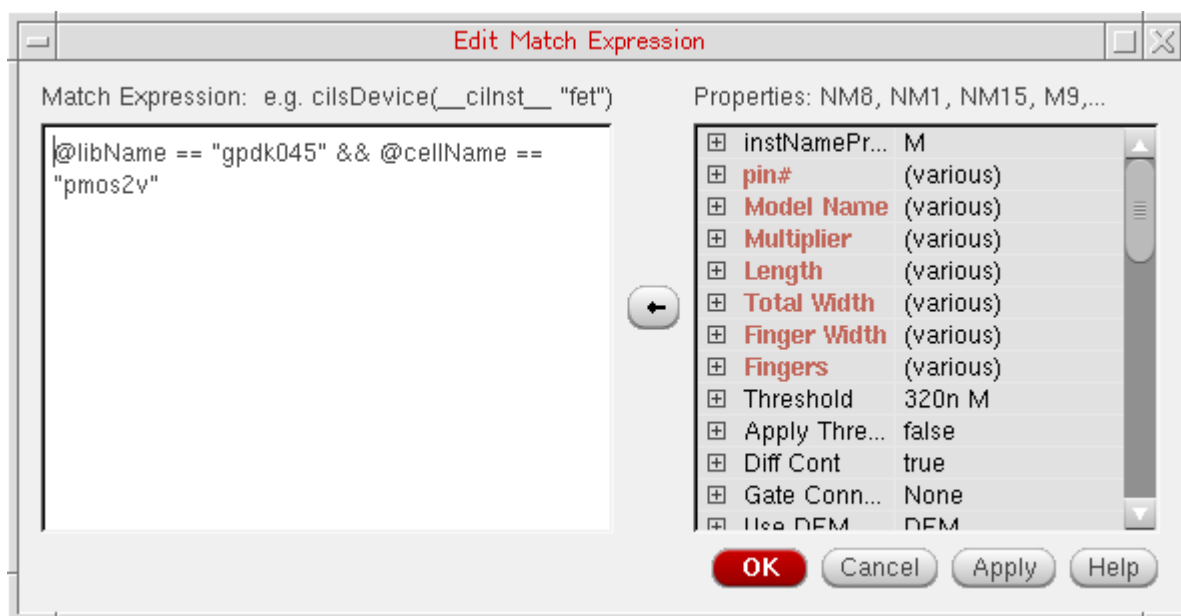


Figure 2-5 The Edit Match Expression form

Edit Terminal Name Expression Form

You can use the Edit Terminal Name Expression form to edit terminal expressions and define connectivity between repeatable instances.

Example of How to Capture A Repeatable Structure Utilizing the Edit Terminal Name Expression Form

The following example describes how to capture a resistor network that compromises of a number of serial resistors (R_0 , R_1 , R_2 , and so on).

1. Only the first two resistors in such a chain require to be selected in the design canvas (for example, R_0 and R_1).

Note: R_0 will be considered to be the root part of the structure and R_1 considered to be the repeatable part of that chain.

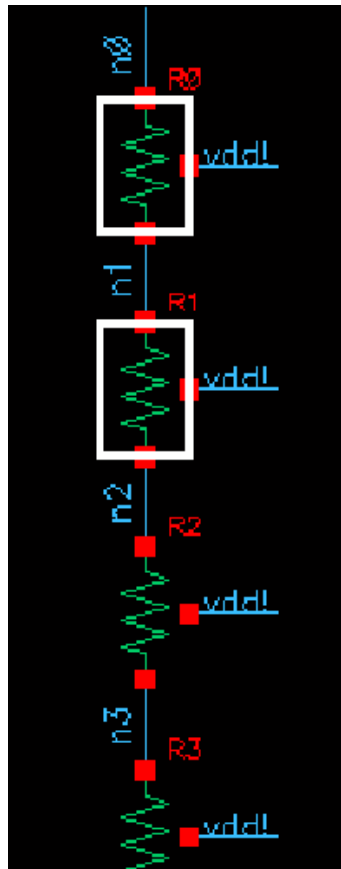


Figure 2-6 Selecting Resistors R_0 and R_1 in the Design Canvas

2. Click the *Capture Structure* option in the *Circuit Prospector* toolbar.

The Capture Circuit Structure form is displayed.

Name: Serial Resistors

Type:

Description:

Expression (insts, nets, pins) e.g. zeroop(length(setof(inst insts inst->orient != car(insts)->orient)))

Objects	Match Express	Repeatable
Insts		
R0	@libName ...	☐
B(vdd!)	ciGetDevic...	
MINUS(n1)	ciGetDevic...	
PLUS(n0)	ciGetDevic...	
R1	@libName ...	☒
B(vdd!)	ciGetDevic...	ExistingNet:vdd!
MINUS(n2)	ciGetDevic...	DontCare
PLUS(n1)	ciGetDevic...	PreviousInstTerm:MINUS
Nets		
Pins		
Constraints		

Edit Expression

Update From Selected

Recapture Constraints

Delete

OK Cancel Apply Help

Figure 2-7 The Capture Circuit Structure Form with Instances R0 and R1 Expanded

3. Ensure that the R0 and R1 instances are expanded and that the *Repeatable* check box for R1 is selected (as R1 is to be repeatable).

The expansion of the R1 instance will display the terminals of that instance and the nets that they are currently connected to.

Each terminal can now be selected, and its respective expression edited in the Edit Terminal Name Expression form. That expression will define how the connectivity between the repeatable instance and the root, or previous, instance must be set.

4. Select a terminal to be edited.
5. Click the *Edit Expression* button.

The Edit Terminal Name Expression form is displayed.

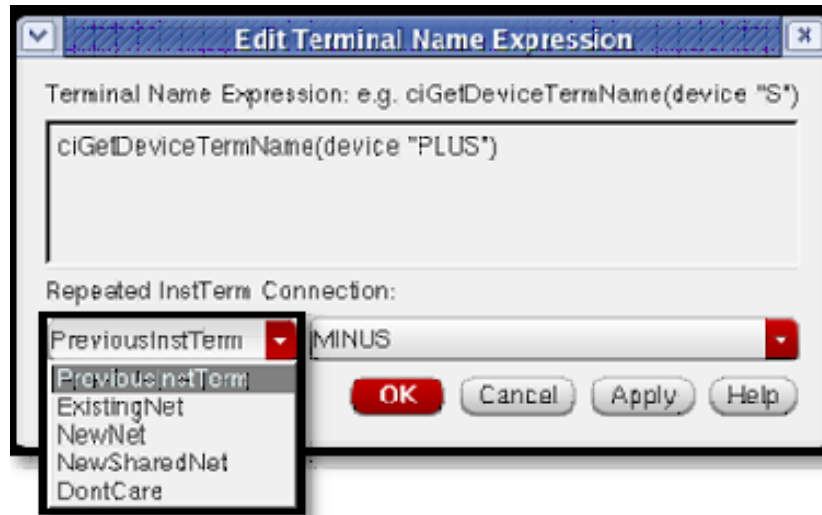


Figure 2-8 The Edit Terminal Name Expression Form

6. In the Edit Terminal Name Expression form you could now, for example, define that:

- ☐ the PLUS terminal of R_n be connected to the MINUS terminal of R_{n-1} (that is, *PreviousInstTerm: MINUS*).
- ☐ the MINUS terminal of R_n must be connected to a *NewNet* (a net that will not be connected to any other device terminal in the structure) or *DontCare* net (may or may not be connected to an existing device terminal in the structure).

Note: For more information on the *DontCare* option, see Figure 2-10 [DontCare Repeated InstTerm Connection](#).

- ☐ the B terminal must be connected to an *ExistingNet* called `vdd!`.
- ☐ the B terminal must be connected to a *NewSharedNet*, meaning that two device terminals sharing the same net should share a new repeated net when those devices are repeated.

Note: For more information on the *NewShareNet* option, see Figure 2-11 [NewShareNet Repeated InstTerm Connection](#).

7. When all expressions have been correctly defined, ensure that a *Name* has been entered in the Capture Circuit Structure form.

8. Click the *OK* button in the Capture Structure form and the new finder will automatically find all chains of serial resistors in the current schematic.

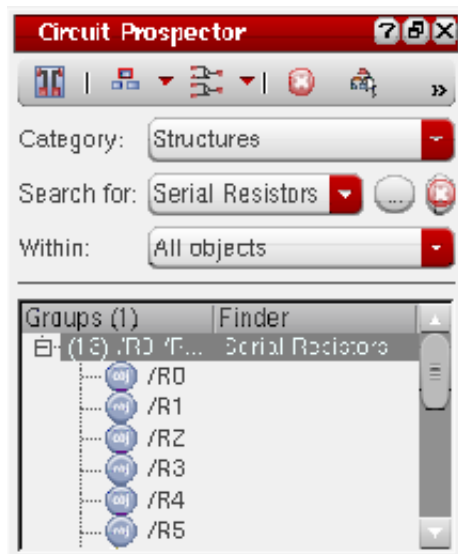


Figure 2-9 The Circuit Prospector Assistant Displaying the Capture of a Repeatable Structure (R0-R5)

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

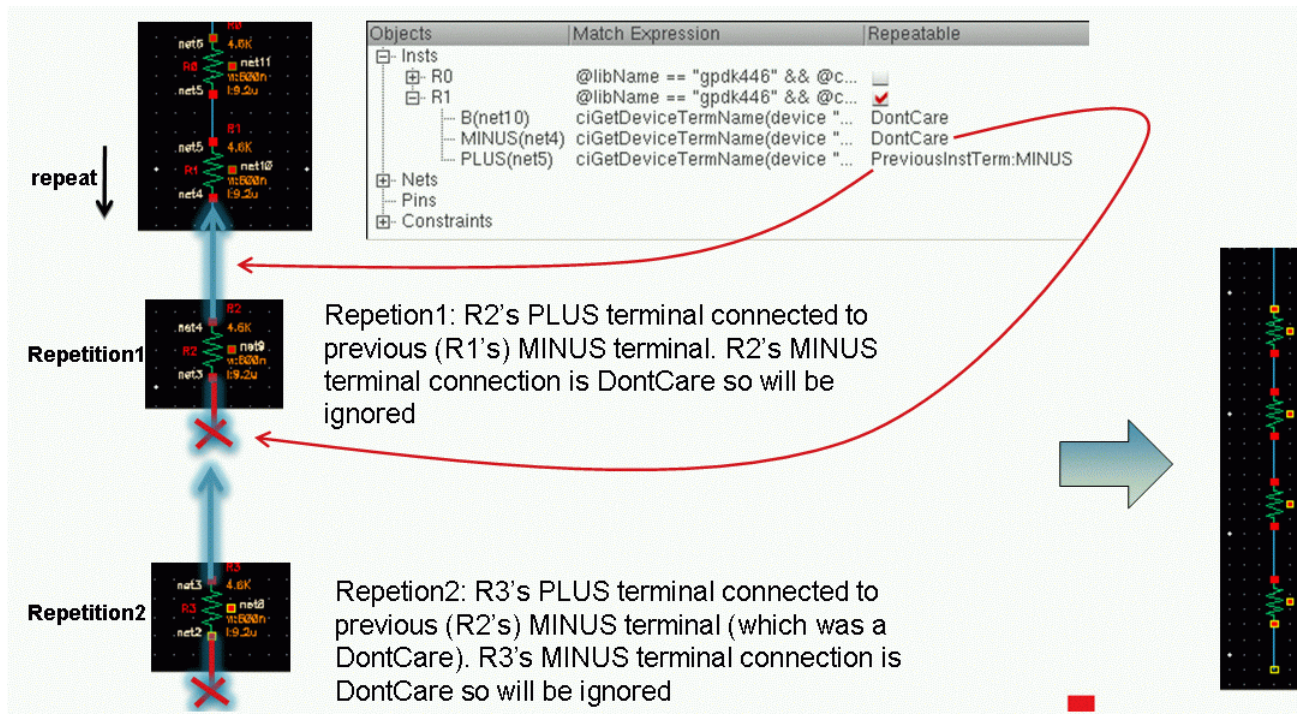


Figure 2-10 DontCare Repeated InstTerm Connection

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

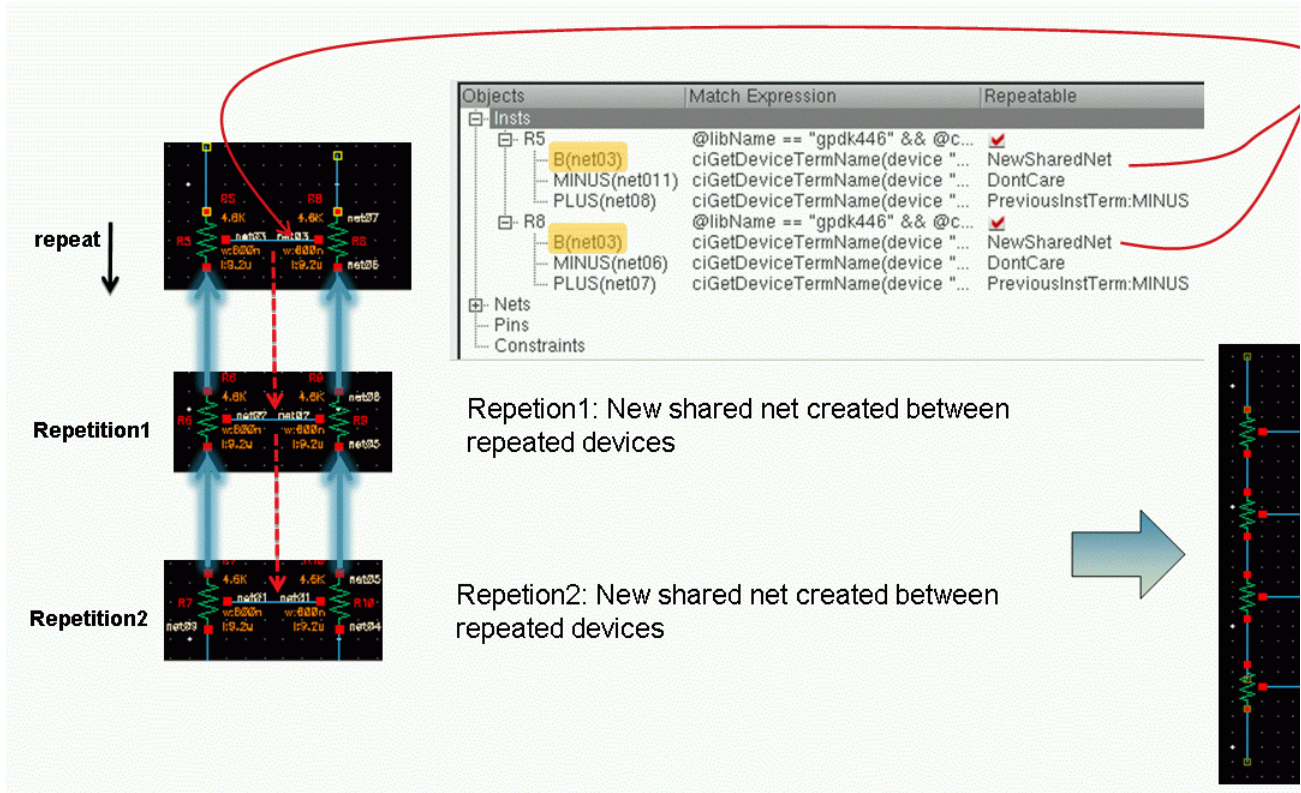


Figure 2-11 NewShareNet Repeated InstTerm Connection

Customization of the Circuit Prospector

The following table summarizes the current categories, finders, and default (associated) constraints that are available in the *Circuit Prospector*.

For more information see:

■ [Categories](#)

■ [Finders](#)

Category	Finder	Default Constraint Generator
Rapid Analog Prototype	MOS Cascode	Module (Cascode MOS Transistors)
	MOS Cascoded Current Mirror	Module (Cascoded Current Mirror)
	MOS Cascoded Current Mirror2	Module (Cascoded Current Mirror)
	MOS Current Mirror	Module (Current Mirror)
	MOS Differential Pair	Module (Diff Pair)
	MOS Differential Pair - Cross Coupled	Module (Diff Pair)
	Passive Arrays	Module (Passive Device Array)
	Active Same Cell Large mfactor	Module (Large mfactor)
	Symmetric Instance Pairs - By Connectivity	Symmetry (default axis)
	Capacitor Cluster	Cluster for Standard Cells
	Vertical Orientation	Orientation Vertical
	Negative Supply	Negative Supply Route Priority
	Positive Supply	Positive Supply Route Priority
	Nets (Symmetry By Connectivity)	Symmetry (default axis)
	Pins (Symmetry By Connectivity)	Symmetry (default axis)
	Top Pins (Alignment)	Alignment for Top Pins
	Bottom Pins (Alignment)	Alignment for Bottom Pins
	Left Pins (Alignment)	Alignment for Left Pins
	Right Pins (Alignment)	Alignment for Right Pins
	Enforce Precedence	Enforce Modgen Symmetry Precedence
	ALL	Matched Parameters
Structures	MOS Cascode	Module (Cascode MOS Transistors)

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

	MOS Current Mirror	Module (Current Mirror)
	MOS Cascoded Current Mirror	Module (Cascoded Current Mirror)
	MOS Cascoded Current Mirror2	Module (Cascoded Current Mirror)
	MOS Transmission Gate	Group + Relative Orientation + Vertical Alignment
	MOS Differential Pair	Module (Diff Pair)
	MOS Differential Pair - Cross Coupled	Module (Diff Pair)
	MOS Parallel	Group + Relative Orientation + Horizontal Alignment
	MOS Active Load	MOS Active Load - Matched + Symmetry + Distance
	MOS Inverter	Distance + Matched Orientation + Vertical Alignment
	Symmetric Instance Pairs - By Pre-selection	Symmetry + Matched
	Symmetric Instance Pairs - By Connectivity	Symmetry (default axis)
	Self Symmetric Instances - By Connectivity	Symmetry (default axis)
	Placement Path	Placement Path
	Series Resistor Array	SeriesResistorArray
	Parallel Net Resistor Array	ParallelResistorArray
	Parallel Resistor Array (Length)	ParallelResistorArray
	Block Resistor Array	BlockResistorArray
	ALL	Matched Parameters
<u>Devices</u>	Active Same Cell	Same Cell Name - Group + Relative Orientation
	Active Same Cell and Size	Group + Relative Orientation + Matched Parameters
	Active Same Cell and Finger Width	Matched Parameters for Same Finger Width
	Active Common Gate	Common Gate - Group + Relative Orientation
	Active Same Cell and Common Gate	Common Gate - Group + Relative Orientation
	Active Same Cell and Common Bulk	Cluster for Devices with common bulk
	Active Same Cell, Common Gate and Bulk	Common Gate - Group + Relative Orientation
	Passive Same Cell	Same Cell Name - Group + Relative Orientation
	Passive Same Cell and Size	Group + Relative Orientation + Matched Parameters
	ALL	Matched Parameters
<u>Nets</u>	Supply Nets	Supply Route Priority
	Non-Supply Nets	Non Supply Route Priority
	Symmetric Nets	Net Symmetry
	Bus and Bundle	Bus for Nets

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

*default property generator	Registered (sigType)	*sigType for registered nets
	Same sigType (none)	<none>
	ALL	Matched Parameters
<u>Pins</u>	Pins (Symmetry By Connectivity)	Symmetry for Pins
	Top Pins (Alignment)	Alignment for Top Pins
	Bottom Pins (Alignment)	Alignment for Bottom Pins
	Left Pins (Alignment)	Alignment for Left Pins
	Right Pins (Alignment)	Alignment for Right Pins
	Supply Pins (Alignment)	Alignment for Top Pins
	Input Pins (Alignment)	Alignment for Left Pins
	Output Pins (Alignment)	Alignment for Right Pins
	ALL	Matched Parameters
<u>Inst Terms</u>	InstTerms by Instance	None
	InstTerms by Name	None
	InstTerms by Net	None
	ALL	Matched Parameters
<u>Properties</u>	Registered Nets (sigType)	*sigType for registered nets
*default property generator	Same sigType Nets (none)	*<none>
*default property generator	fets (variantInfo)	*variantInfo for FETs
*default property generator	Same variantInfo Devices	*<none>
*default property generator	pfets (placerControlledWell)	*placerControlledWell
*default property generator	Same placerControlledWellDevices)	*<none>
	ALL	Matched Parameters
<u>Electrical Constraints</u>	Symmetric Pairs (Correlation)	Correlation
	Same Cell, Size, Bulk Devices (Matched Parameters)	Matched Parameters for Same Cell Size and Bulk

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

	ALL	Matched Parameters
<u>Placement Constraints</u>	Common Bulk (Cluster)	Cluster for Devices with common bulk
	Digital Cells (Cluster)	Cluster for Standard Cells
	Same Cell, Size, Bulk Devices (Modgen)	Modgen for Same Cell Size and Bulk
	Clusters (Cluster Boundary)	Cluster Boundary for Cluster
	Clusters (Guard Ring)	Guard Ring for Cluster
	Modgens (Guard Ring)	Guard Ring for modgen
	Same Device Type (Orientation)	Orientation per device type
	Modgens (Orientation)	Orientation for modgen
	Inverters (Alignment)	Alignment for Inverters
	Inverters (Distance)	Distance for Inverters
	ALL	Matched Parameters
<u>Routing Constraints</u>	Supply Nets (Net Priority)	Supply Route Priority
	Bus and Bundle (Bus)	Bus for Nets
	Bus Bits	Bus for Bus Signal Bits
	ALL	Matched Parameters
<u>Symmetries</u>	Instances (Symmetry By Pre-selection)	Symmetry
	Instances (Symmetry By Connectivity)	Symmetry for Instance
	Nets (Symmetry By Connectivity)	Symmetry for Nets
	Pins (Symmetry By Connectivity)	Symmetry for Pins
	ALL	Matched Parameters
<u>Matched Parameters</u>	Symmetric Pairs (Matched Parameters)	Matched Parameters for Symmetric Instances
	Same Cell, Size, Bulk Devices (Matched Parameters)	Matched Parameters for Same Cell Size and Bulk
	Same Cell, Finger Width MOS (Matched Parameters)	Matched Parameters for Same Finger Width
	Current Mirror (Matched Parameters)	Matched Parameters for Current Mirror
	ALL	Matched Parameters

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

<u>Clusters</u>	Common Bulk (Cluster)	Cluster for Devices with common bulk
	Same Cell and Size (Cluster)	Cluster for Devices with same name and size
	Same Cell, Finger Width MOS (Cluster)	Cluster for Devices with same finger width
	Digital Cells (Cluster)	Cluster for Standard Cells
	ALL	Matched Parameters
<u>Modgens</u>	Same Cell, Size, Bulk Devices (Modgen)	Modgen for Same Cell Size and Bulk
	Digital Cells (Modgen)	Modgen for Standard Cells
	ALL	Matched Parameters
<u>Orientations</u>	Orientation as in Schematic	Same Orientation
	Same Device Type (Orientation)	Orientation per Device Type
	Modgens (Orientation)	Orientation for modgen
	ALL	Matched Parameters
<u>Analog Device Structures</u>	APR Device Group from DI	Device Group Generator
	APR Differential Pair	Device Group Generator
	APR Differential Pair - Cross Coupled	Device Group Generator
	APR Current Mirror	Device Group Generator
	APR Cascode Current Mirror	Device Group Generator
	APR Cascode Series Current Mirror	Device Group Generator
	APR Cascode	Device Group Generator
	APR Active Same Size Common Gate	Device Group Generator
	APR Active Same Size Common Gate and Source	Device Group Generator
	APR Active Same Size Common Source	Device Group Generator
	APR Active Same Cell M-Factored Device	Device Group Generator
	APR Active Same Cell Stacked Device	Device Group Generator
	APR Active Same Cell Iterated Device	Device Group Generator
	APR Passive Same Cell and Size	Device Group Generator
	APR User Defined Group	Device Group Generator
	APR Instances (Symmetry By Connectivity)	Device Group Generator

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

APR Instances (Symmetry By Connectivity with common source or drain)	Device Group Generator
APR Nets (Symmetry By Connectivity)	Device Group Generator
APR Pins (Symmetry By Connectivity)	Device Group Generator

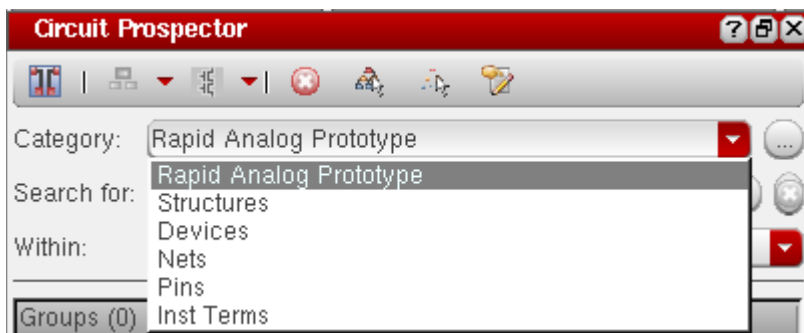
Categories

Note: See also [Creating and Editing Categories](#) in this book and [Creating New Categories](#) in the [Virtuoso Unified Custom Constraints Configuration Guide](#).

Categories are groups of related [Finders](#) (search algorithms) that are used to store structural definitions.

[Finders](#) search for devices/pins/nets that match certain customizable search criteria, for example common circuit structures, related devices (such as same size, area, or type), devices with particular characteristics or connectivity, symmetrical devices/nets/pins, or devices with particular properties. The group of finders associated with a category represent a grouping of related finders, for example all fet related finders and/or a constraint creation flow (for example, create groups, find symmetries, and find structures).

By default, the *Circuit Prospector* provides you with a set of pre-defined categories that can be listed by clicking the *Category* drop-down list box (as shown below).



The following pre-defined categories are displayed in the *Category* drop-down list box by default:

- [Rapid Analog Prototype](#)
- [Structures](#)
- [Devices](#)
- [Nets](#)
- [Pins](#)
- [Inst Terms](#)

The following pre-defined categories are also available for you, but are disabled by default:

- [Properties](#)

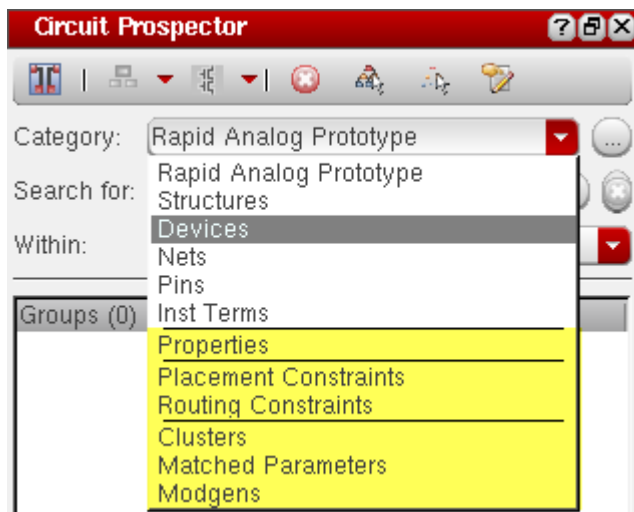
Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

- Electrical Constraints
- Placement Constraints
- Routing Constraints
- Clusters
- Matched Parameters
- Modgens
- Orientations
- Symmetries
- Analog Device Structures

To use any of these disabled pre-defined categories, you need to enable them by using the `ciEnableAssistant` SKILL function. The categories thus enabled start showing in the *Category* drop-down list box after you close the current Virtuoso Schematic Editor XL window and launch it again from Virtuoso. For example, if you run the following SKILL commands in the CIW, the *Category* drop-down list box is updated as highlighted in the figure below:

```
ciEnableAssistant("Properties" t)
ciEnableAssistant("Placement Constraints" t)
ciEnableAssistant("Routing Constraints" t)
ciEnableAssistant("Clusters" t)
ciEnableAssistant("Matched Parameters" t)
ciEnableAssistant("Modgens" t)
```

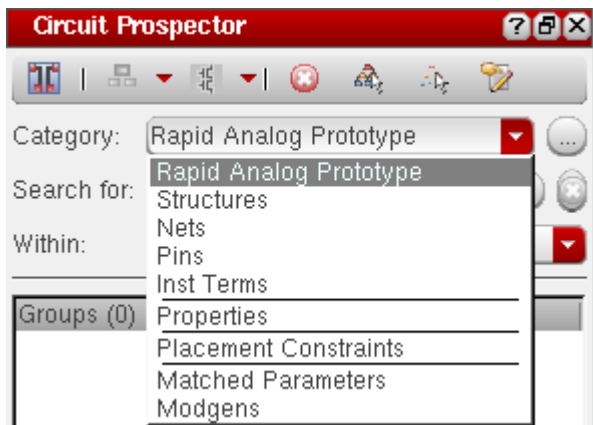


Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

The `ciEnableAssistant` SKILL function can also be used to disable any pre-defined category (including the ones enabled by default). For example, when you run the following SKILL commands in the CIW, the specified categories get disabled and the *Category* drop-down list box gets updated accordingly (refer to the figure below):

```
ciEnableAssistant("Devices" nil)
ciEnableAssistant("Routing Constraints" nil)
ciEnableAssistant("Clusters" nil)
```



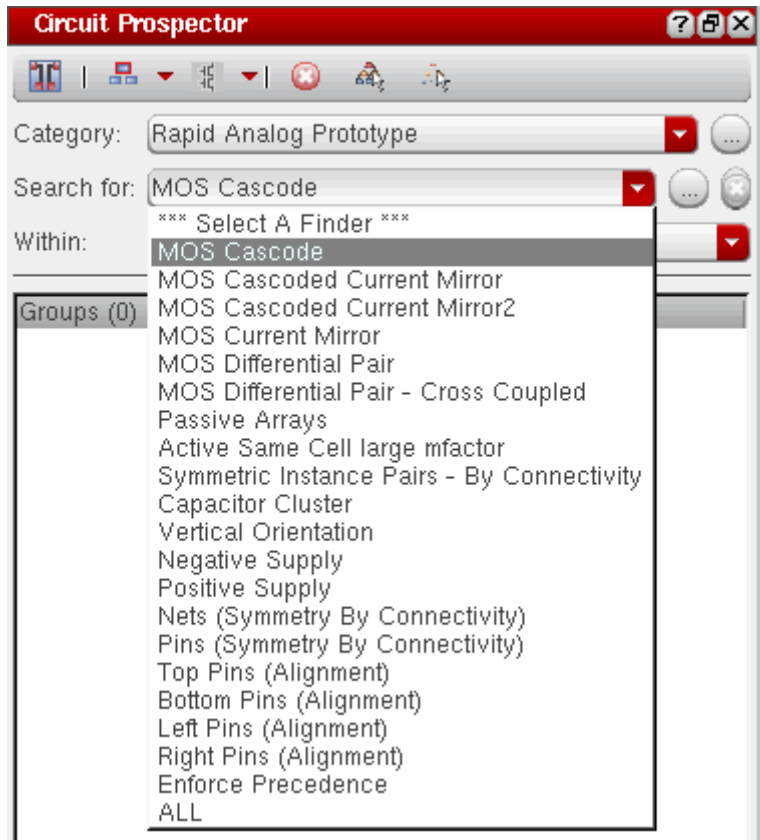
Rapid Analog Prototype

The Rapid Analog Prototype (RAP) category in the Circuit Prospector contains all the finders/generators required to automatically constrain a design with just a few mouse clicks. The resulting constraints result in good initial placement and routing from which layout effects (parasitic and device effects) can be extracted and taken into account during simulation early in the circuit design phase. Associated with the finders are dedicated modgen generators for common structures that automatically generate optimized structure specific modgens. These

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

generators capture layout expertise and allow this expertise to be automatically re-applied to each structure in a design.



In addition, it is fully customizable to allow you to generate modgens to your particular requirements. It also includes template parameters for allowing modgen re-configuration in schematic and layout without the need to bring up the specialized Modgen Editor.

Combining these devices into modgens reduces the number of devices and constraints that the Analog Place and Route tools have to work with leading to faster run times, better results, and more predictability.

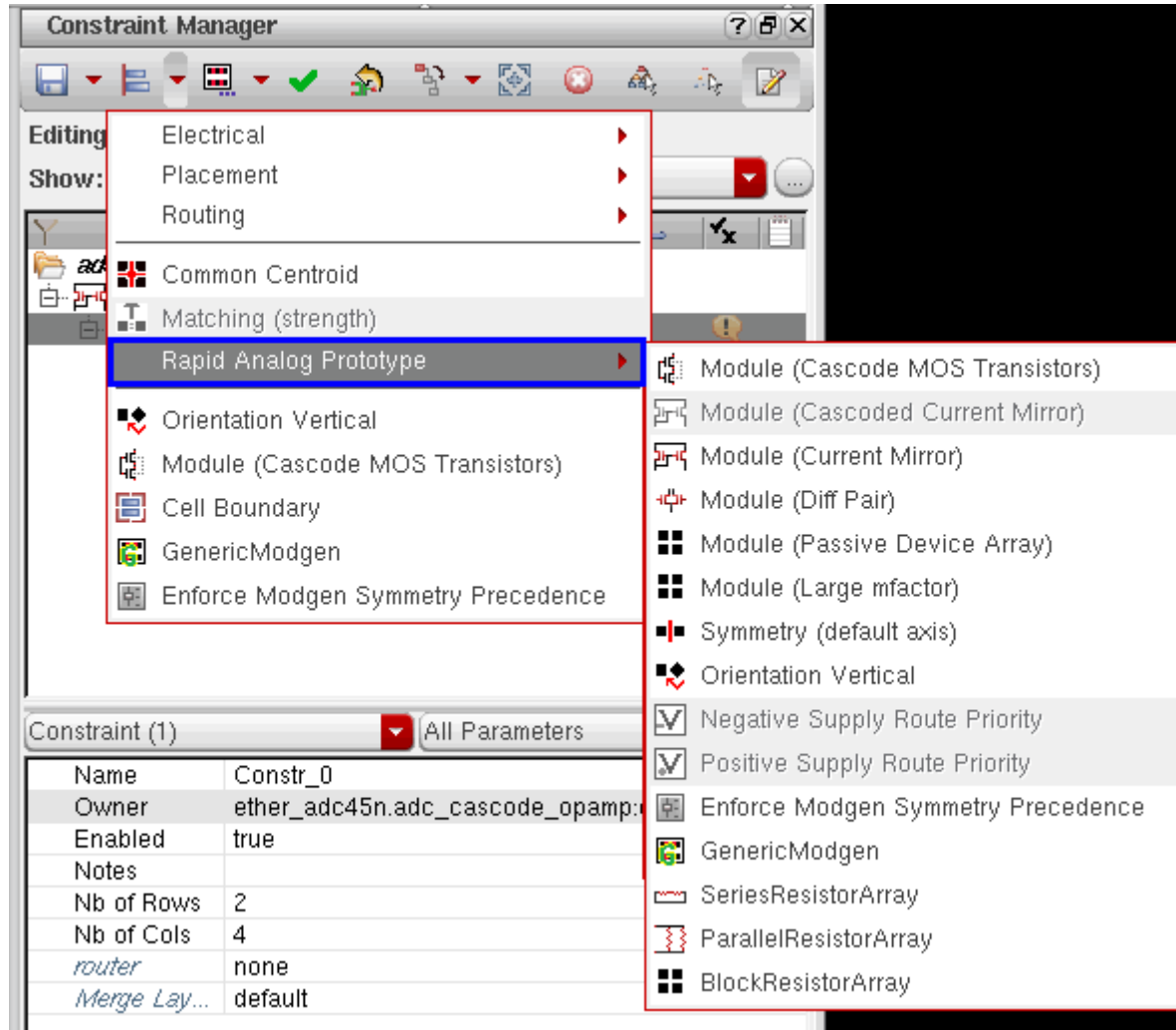
The finders/generators in the RAP category will generate all the constraints required to achieve a good initial (prototype) layout.

Most generators in the RAP category are also available in the Constraint Manager assistant under the *Rapid Analog Prototype* sub-menu (the image below shows the available

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

generators.) This allows you to manually select the required structures on the schematic or layout canvas and run the required RAP generator on them.



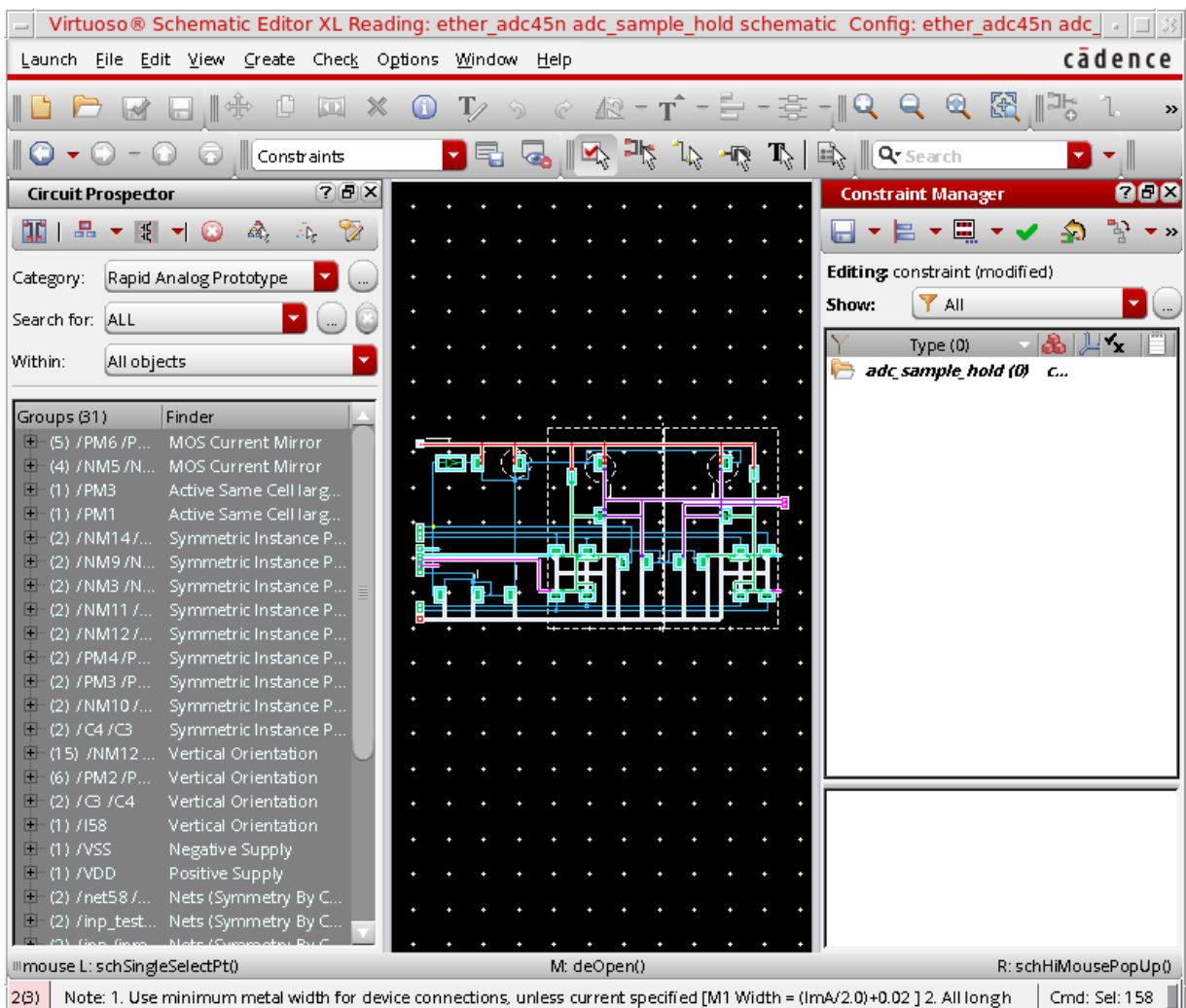
Prototyping Flow using the RAP Finders

The RAP finder-based prototyping flow requires you to perform the following simple steps.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

1. Select *ALL* from the *Search for* drop-down list box to run all the finders in the RAP category.



2. Then, run the default constraint generator for the RAP results. Constraint Generator dialogs will popup for each RAP constraint generator allowing you to configure the generator parameters for the associated set of devices. The device names are displayed in the title of the dialog box.

The dialog box normally has four buttons: *OK*, *Cancel*, *Defaults*, *Apply*, and *Help*.

When you click the *OK* button, the specified values get saved as the default values for the consecutive access of the same dialog box. For example, you created a MOS Current Mirror constraint, changed the setting of the *Abut All* check box, and clicked *OK* on the dialog box. Then, when you create a MOS Current Mirror constraint again on a different set of instances, the dialog box remembers the parameter values saved last. If the

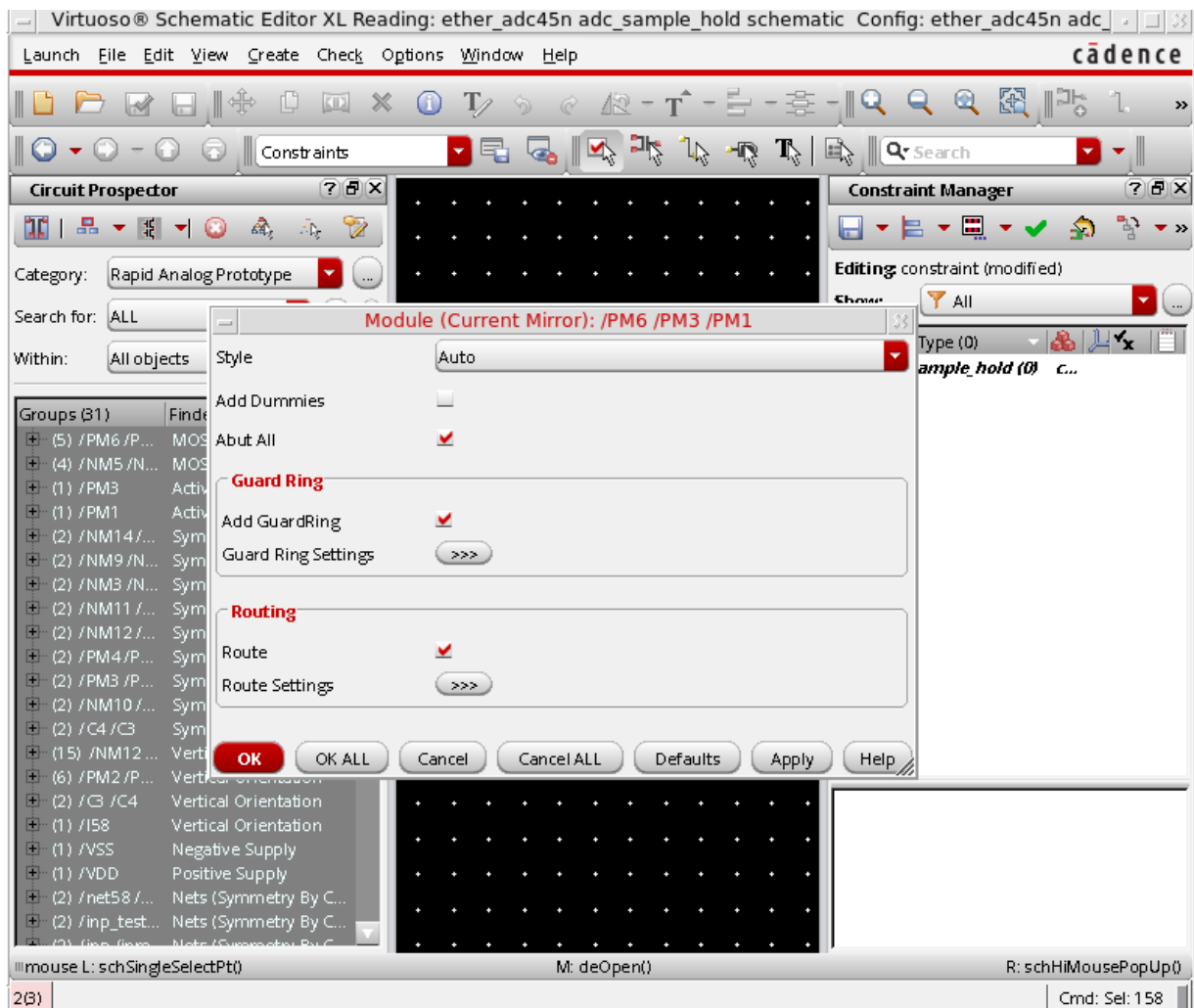
Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

parameter values specified by you are incompatible with the last save, the default ones are set instead.

The *Defaults* button can be used to rollback any user-defined values in the fields to their valid default values.

When you select multiple RAP results from the list and click a constraint generator button, the form provides additional *OK ALL* and *Cancel ALL* buttons.



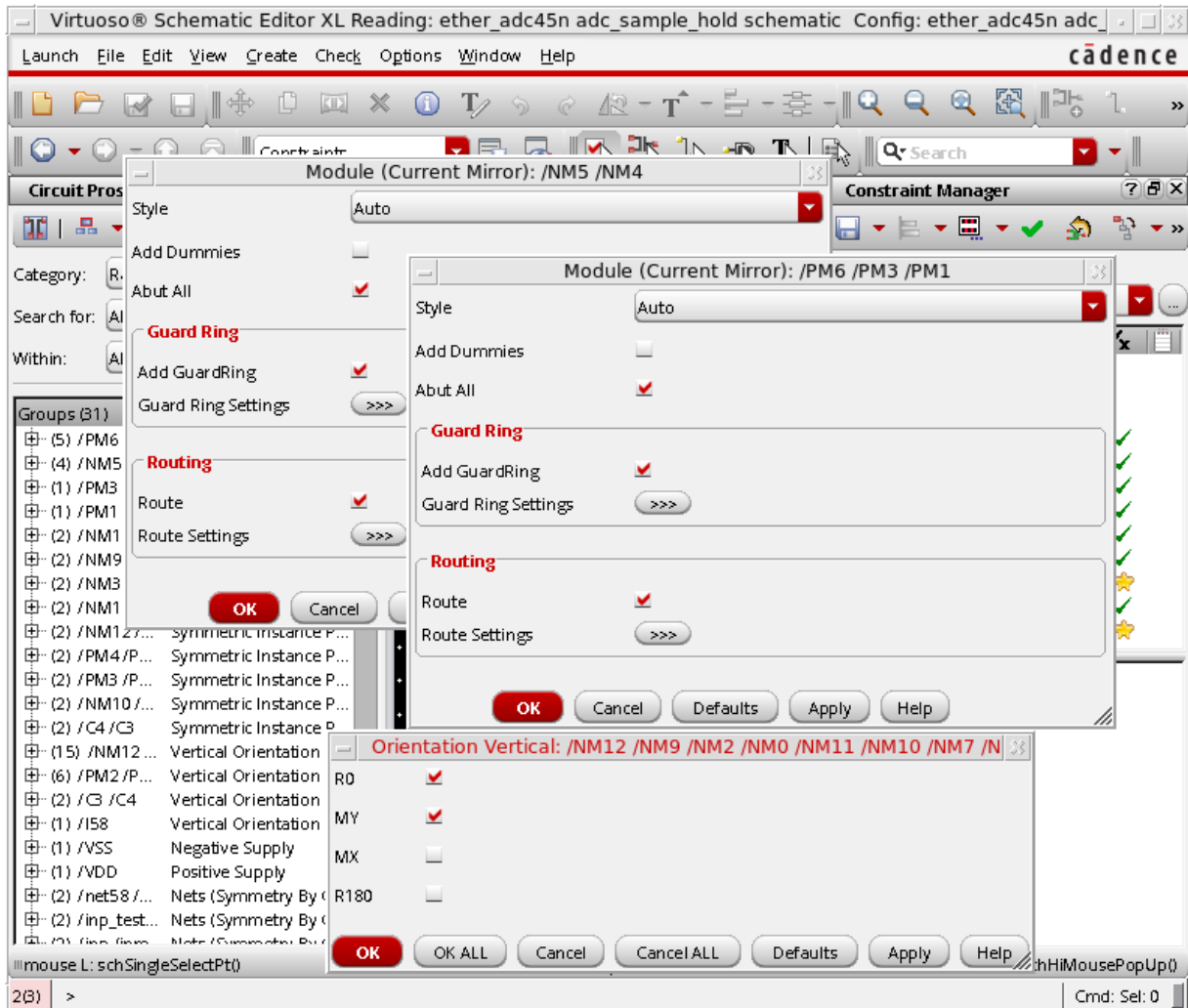
The *OK ALL* button allows you to accept the default values for all subsequent constraint generators and the dialog boxes will not popup for those. Here, the default values depend on the last values set by the generator on a template.

The *Cancel ALL* button cancels the current constraint generator and all subsequent constraint generators.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

If these additional buttons are displayed and you choose to click the *Apply* button, the *OK ALL* and *Cancel ALL* buttons disappear from the dialog box. In addition, different dialog boxes for each selected RAP result become visible for entering the required constraint parameters, as shown in the figure below.

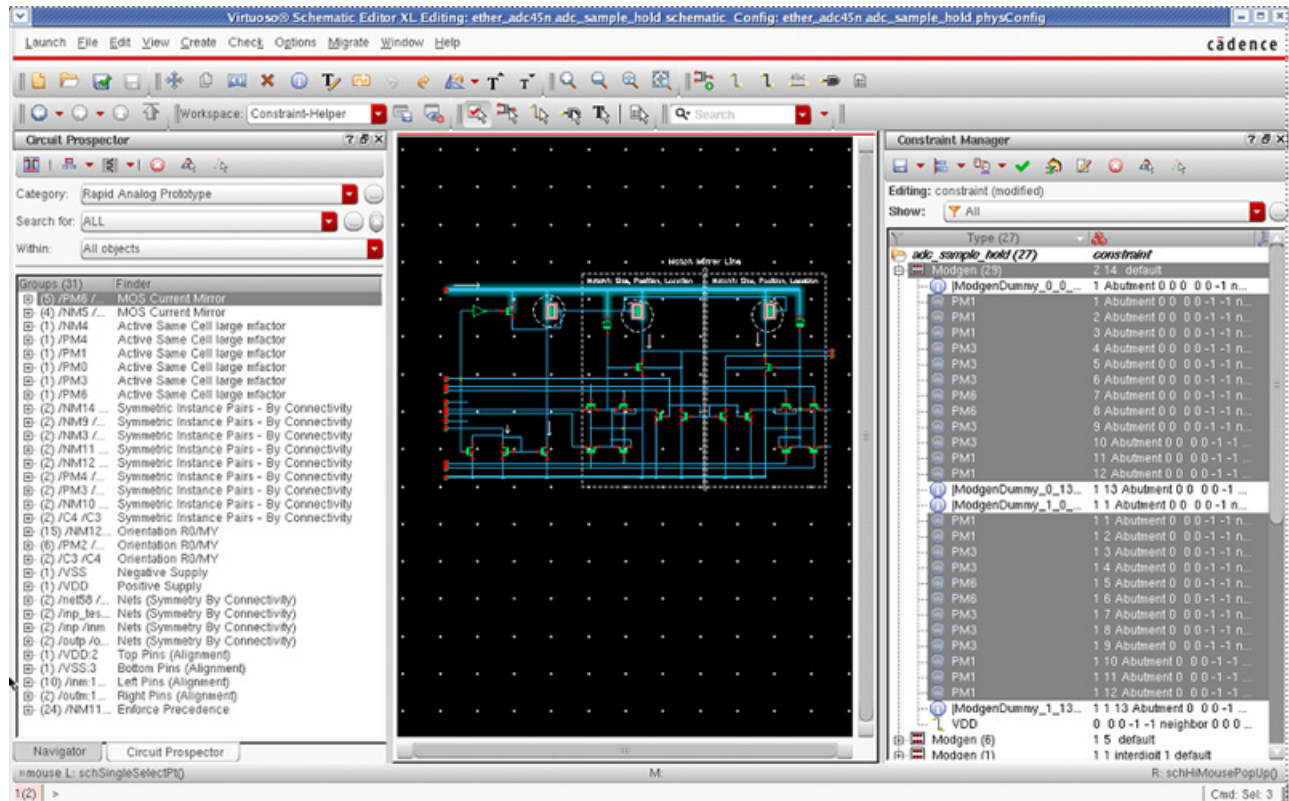


Optimized modgens are created for each type of structure is found, including Differential Pair, Current Mirror, and Large mfactor devices. The modgens capture analog design/layout engineer expertise for each type of modgen being created, for example, Rows/Columns, Patterns, Dummies, Guard Rings, and Body Contacts. In addition, Symmetry constraints are created for symmetrical devices and nets, Rail, and Net Priority, where as, Process Rule Override constraints are created for the supply nets. Vertical

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

Orientation constraints are created on all devices and the pins are aligned on the cell boundary according to their placement in the schematic.

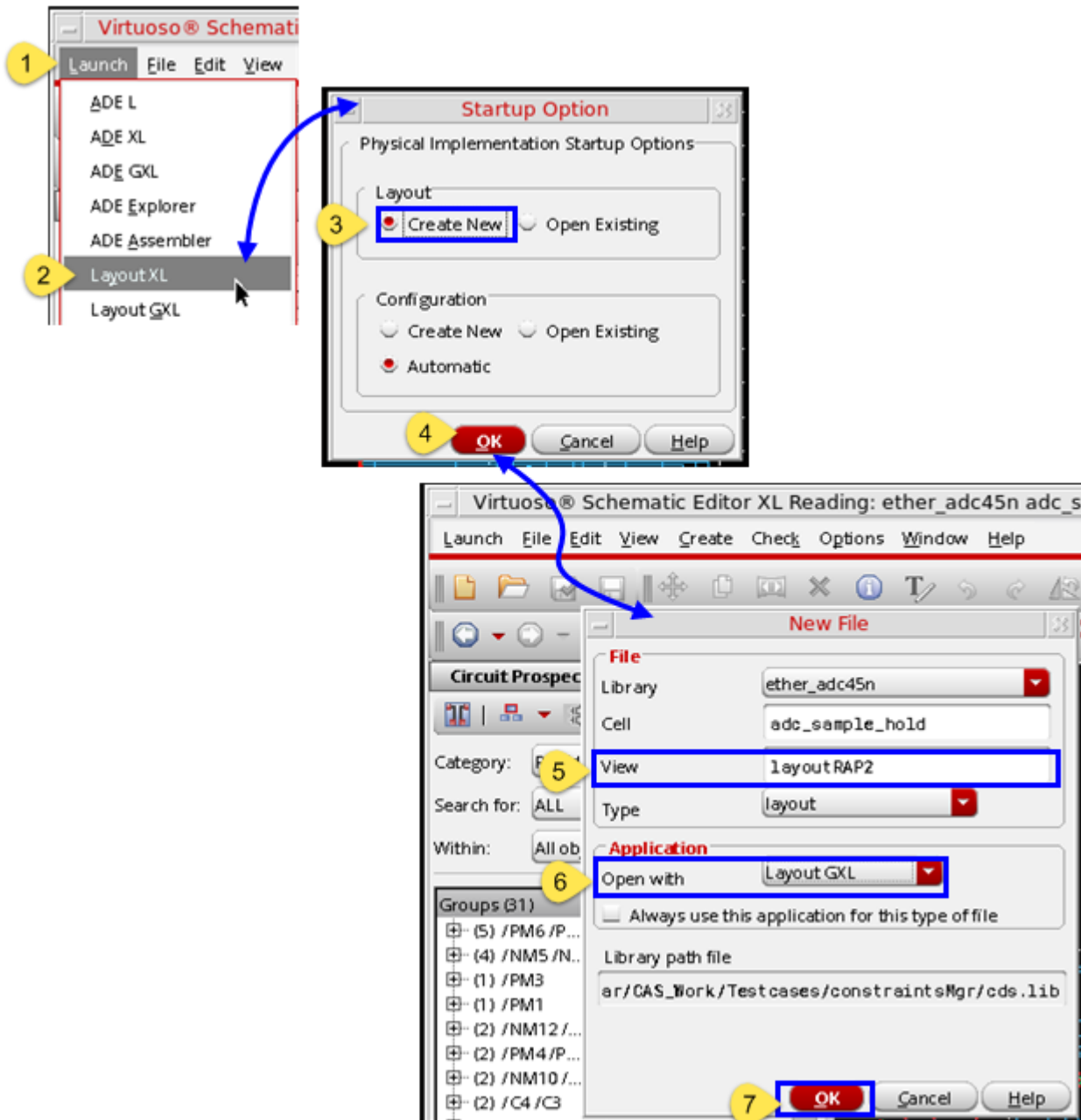


At this point, with modgens created in the schematic it is possible to re-simulate the design taking into account the LDE effects on the devices in the modgens. For more information on LDE, refer to the [Simulating Designs with LDE](#) section in the [Virtuoso ADE Assembler User Guide](#).

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

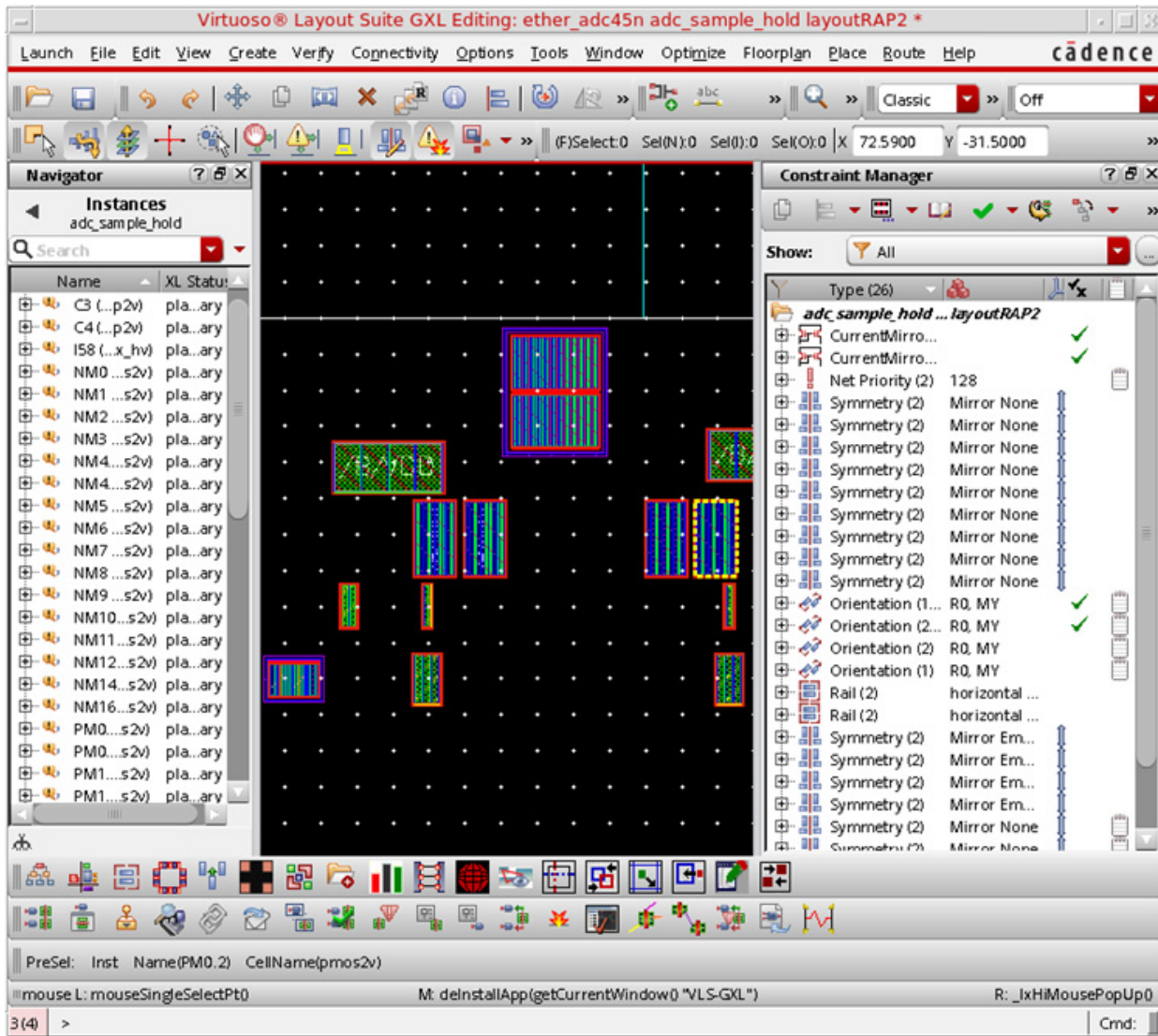
3. Now, click *Launch – Layout XL*, select *Create New* in the *Layout* group box, and click *OK* to open the *New File* form. In this form, specify a new *View* name and *Application* to open with before clicking *OK*. These steps are shown in the figure below.



Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

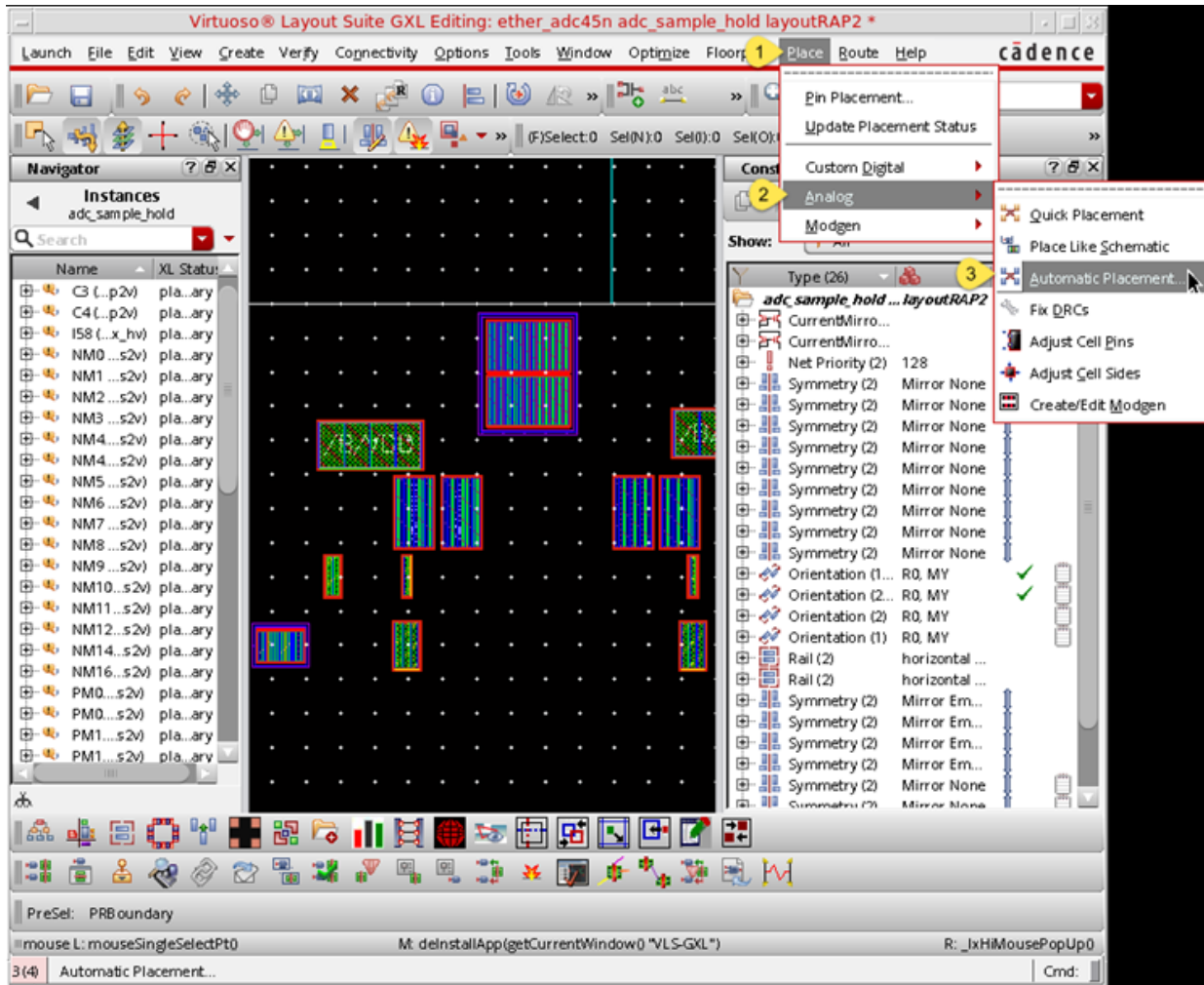
4. To generate layout representations of the schematic design components, select the *Connectivity – Generate – All From Source* option.



Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

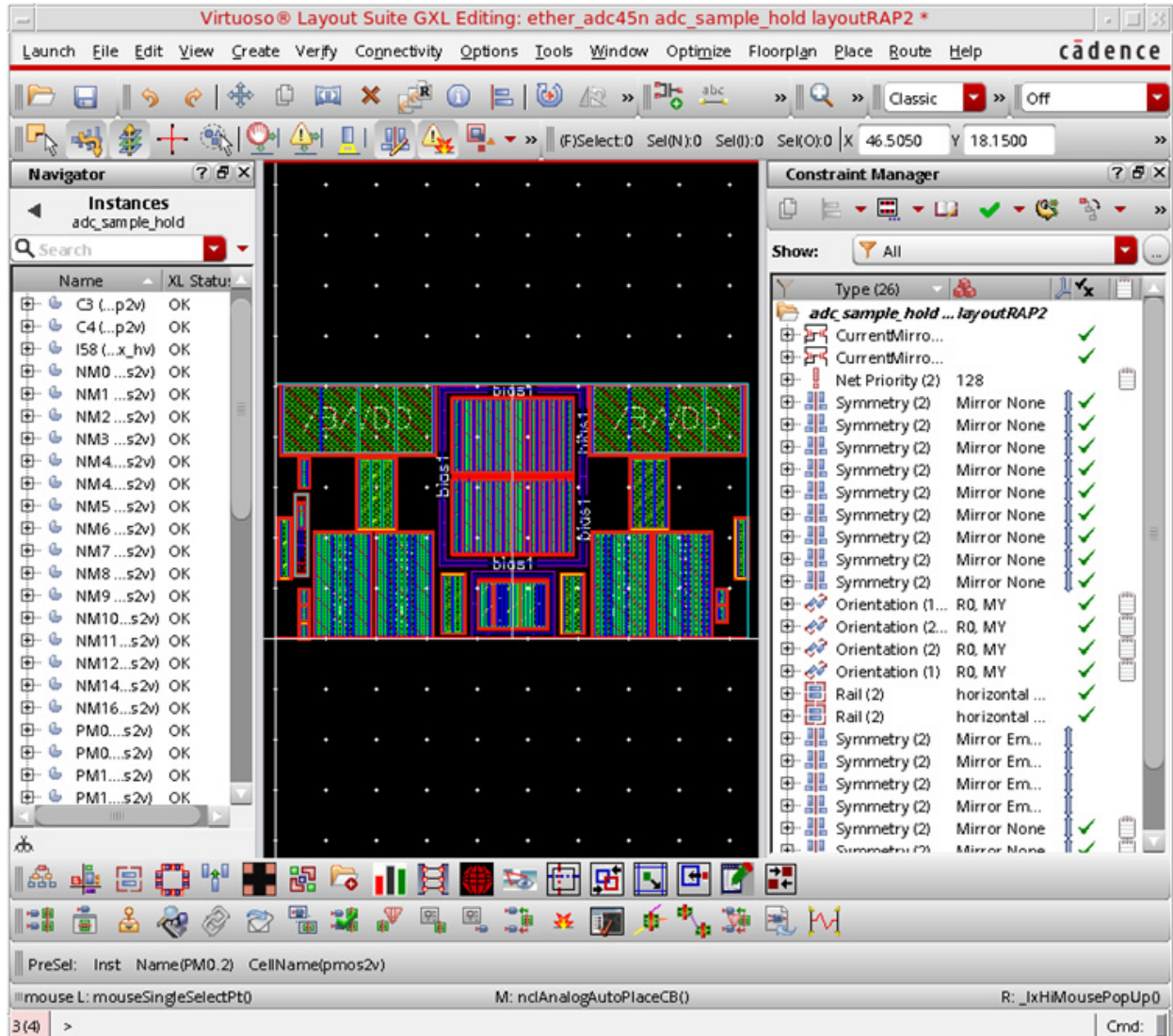
5. (IC6.1.8 Only) Start analog automatic placement by selecting the *Place – Analog – Automatic Placement* option.



Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

The automated placement result is displayed, as shown below.

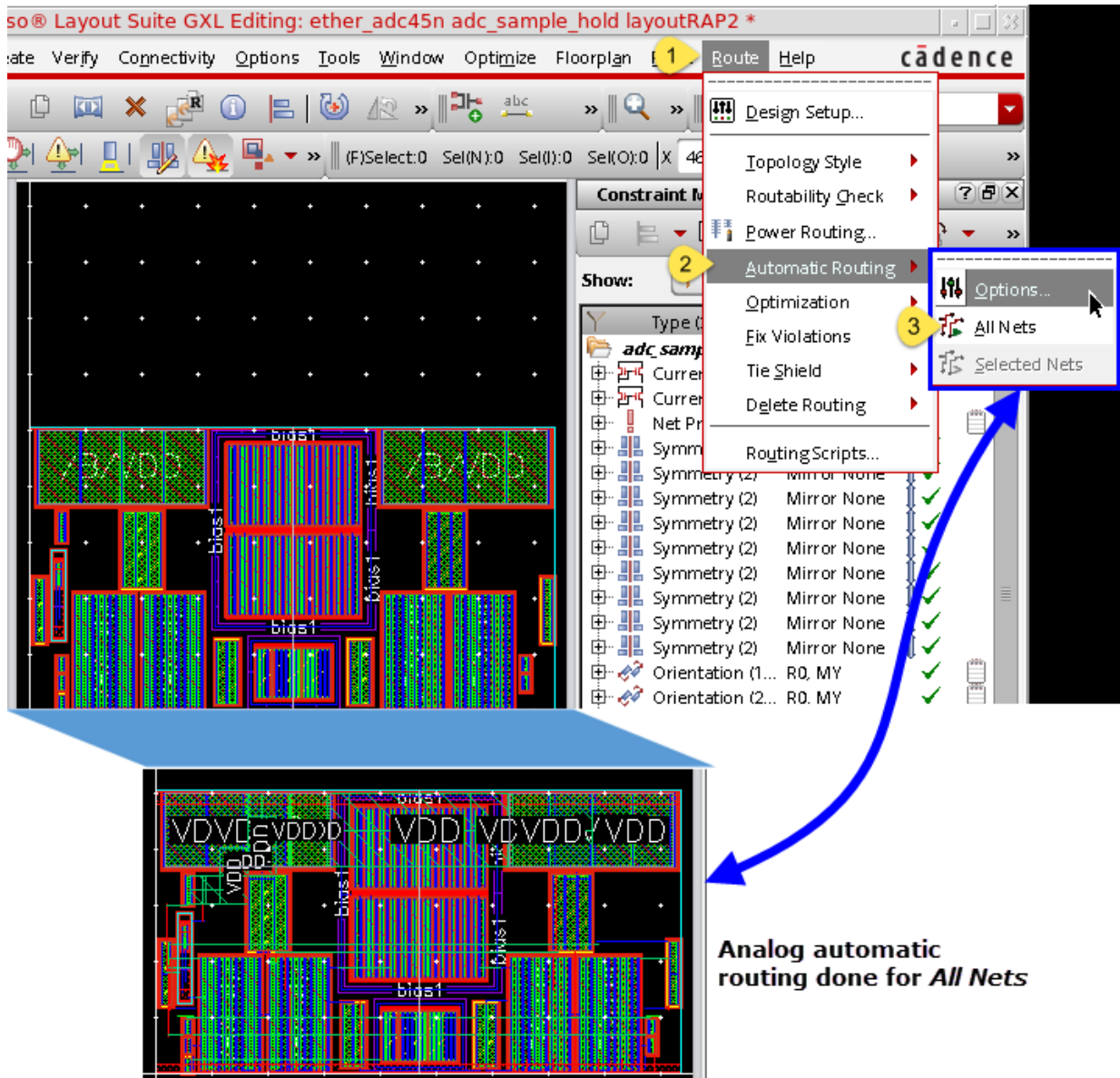


Constraining the critical devices as Modgens not only helps capture design intent more accurately but also leads to better overall placement and faster convergence. At this point, with devices and modgens placed it is possible to re-simulate the design taking into account the LDE effects on the placed devices. For more information on LDE, refer to the [Simulating Designs with LDE](#) section in the *Virtuoso ADE Assembler User Guide*.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

6. Start Analog Automatic Routing by selecting the *Route – Automatic Routing* – any of the given submenu options. The figure below shows how automatic routing is done for all nets when you select the *All Nets* submenu.



7. Run PVS-CV and re-simulate the design with parasitics. For more information, refer to the *An Analog Simulation Flow using PVS LDE* section in the *Virtuoso Parasitic Aware Design User Guide*.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

Template Parameters

By default, all RAP constraint generators that generate modgens creates the modgens within templates. The templates have the same parameters as those specified for the constraint generator that generated the modgen. You can modify the template parameters after the modgen has been created, which will trigger the modgen to be regenerated based on the new parameter settings. It is also possible to modify the template parameters within VSE XL, VLS XL, and the modgen editor.

Module (Current Mirror): /PM6 /PM3 /PM1

Style: Auto

Add Dummies: ☐

Abut All: ☒

Guard Ring

Add GuardRing: ☒

Guard Ring Settings:

Type: ring

Shape: rectangular

Net: bias1

Spacing: 0

MPP: N-Tap(gpd045)

Fluid Guard Ring

Use Fluid: ☐

Device: undefined

Width: 0

Use Min DRC for Spacing: ☐

Routing

Route: ☒

Route Settings:

OK Cancel Defaults Apply Help

Constraint Generator Args

Constraint Manager

Editing: constraint (modified)

Show: All

Type (1)

adc_sample_hot... const...

CurrentMirr...

Modgen ... 212 ...

Guard Ri...

Template (1) All

Name	MatchTemplate0
Created	abhar (today)
Modified	
Checked	
Notes	
Style	Auto
Add Dummies	false
Abut All	true
Add GuardRi...	true
Type	ring
Shape	rectangular
Net	bias1
Spacing	0
MPP	N-Tap(gpd045)
Use Fluid	false
Device	undefined
Width	0
Use Min DR...	false
Route	true
...	...

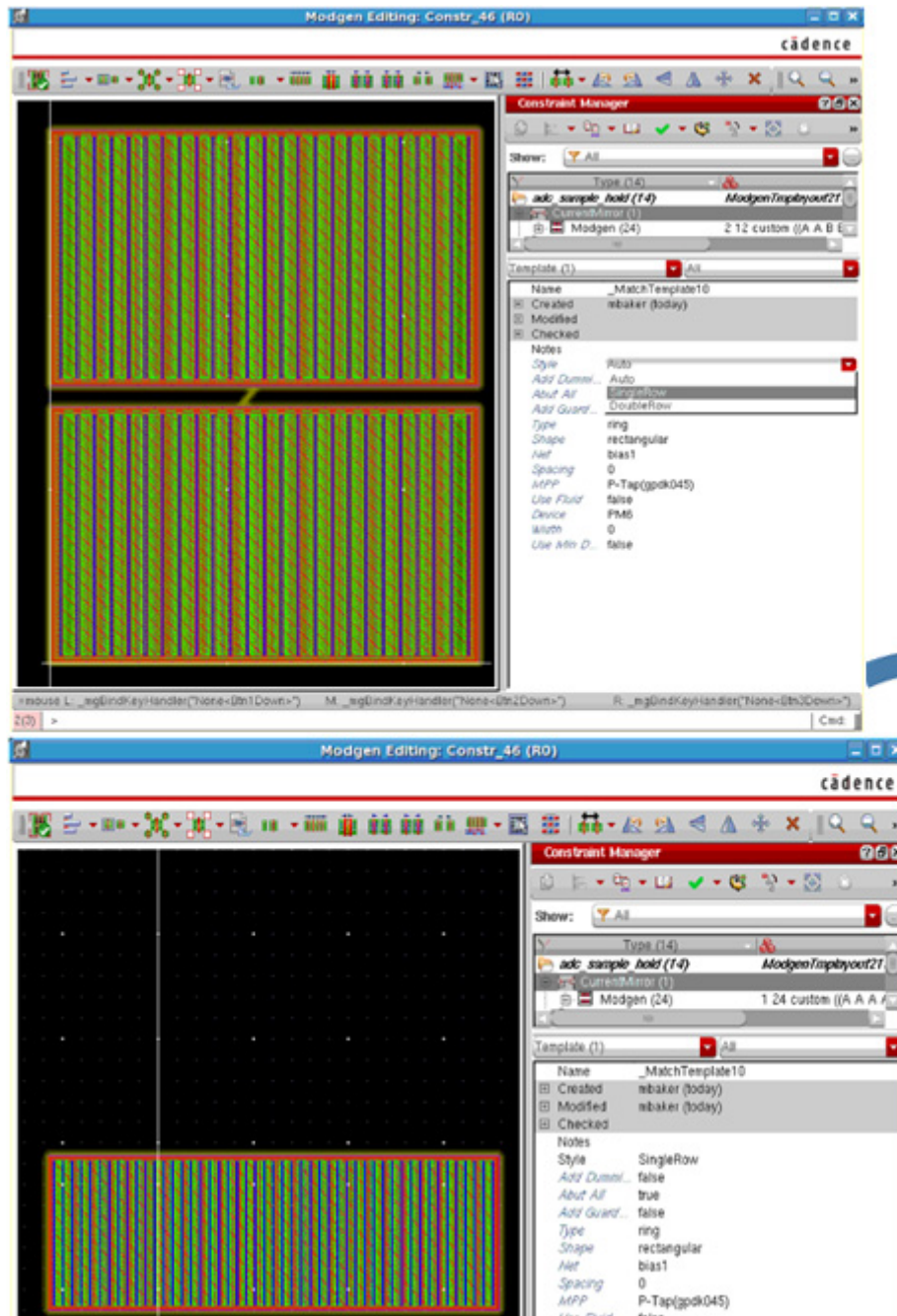
Template Args

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

Note: If the *Add GuardRing* option is selected, the guard ring associated with the modgen will also be added to the template.

Changing the Style template parameter from *DoubleRow* to *SingleRow* re-invokes the Current Mirror constraint generator to regenerate the modgen for the Current Mirror on a single row with the appropriate interdigitation pattern.



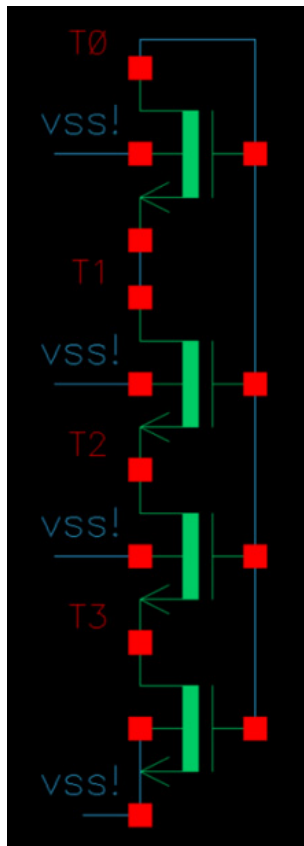
Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

RAP Finders

All RAP [Finders](#) and their associated constraint generators are detailed below.

Devices Finder	Description
<i>MOS Cascode</i>	Lists the following types of MOS transistors: ■ Transistors that share the same gate ■ Transistors that are connected to each other through a common source, as illustrated in the following figure:



When you generate a modgen constraint for the cascoded MOS transistors, all connected transistors are abutted in the layout. This results in an optimum and compact placement of the MOS transistors.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

It will generate patterns similar to the MOS Current Mirror based on the option you select from the *Style* drop-down list box in the Module (Cascode MOS Transistors) form.

As shown in the figure below, the following options are available in the *Style* drop-down list box:

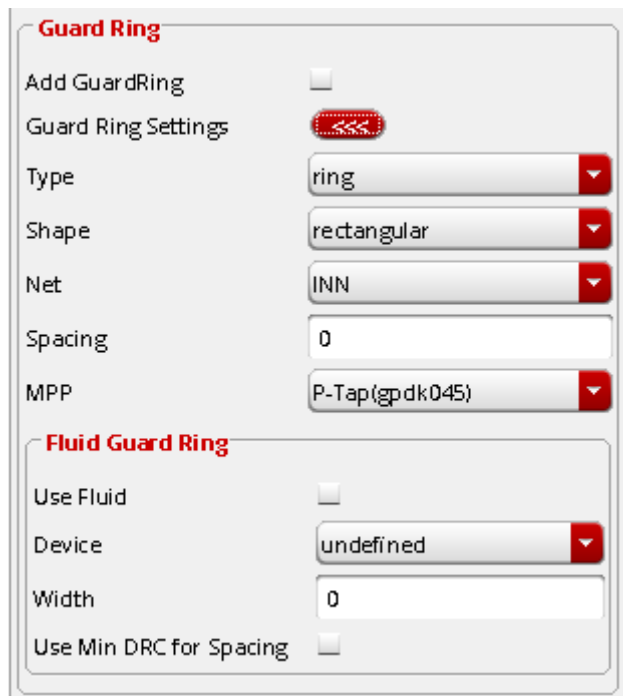


- *Style (Auto)*: Automatically picks the best number of rows based on the aspect ratio of the different possible row counts.
- *Style (SingleRow)*: Sorts the devices by mfactor based on the number of fingers. It places the largest mfactored devices at the two ends of a modgen and the smaller mfactored devices in between them.
- *Style (DoubleRow)*: It is same as the *SingleRow* option, but it places the mfactored devices in two rows.
- *Add Dummies*: Controls whether to add dummies.
- *Abut All*: Controls device abutment. When selected, abutment of devices is enabled.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

- **Guard Ring:** Controls whether a guard ring should be added. Use the *Guard Ring Settings* button to view and set the standard guard ring options shown in the figure below.



The screenshot shows a dialog box titled "Guard Ring" with a red header. It contains two sections: "Guard Ring" and "Fluid Guard Ring".

Guard Ring Section:

- Add GuardRing:** A checkbox that is currently unchecked.
- Guard Ring Settings:** A red button with a double arrow icon.
- Type:** A dropdown menu with "ring" selected.
- Shape:** A dropdown menu with "rectangular" selected.
- Net:** A dropdown menu with "INN" selected.
- Spacing:** A text input field with "0" entered.
- MPP:** A dropdown menu with "P-Tap(gpkg045)" selected.

Fluid Guard Ring Section:

- Use Fluid:** A checkbox that is currently unchecked.
- Device:** A dropdown menu with "undefined" selected.
- Width:** A text input field with "0" entered.
- Use Min DRC for Spacing:** A checkbox that is currently unchecked.

The standard guard ring options available in the *Guard Ring* group box are the same as discussed in the [Adding a Guard Ring Around Generated Modgen](#) section of Chapter 1, "The Constraint Manager Assistant."

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

- **Routing:** Controls whether pin to trunk routing should be enabled. Use the *Route Settings* button to view and set the standard routing options shown in the figure below.

The screenshot shows a dialog box titled "Routing" with a red header. It contains the following settings:

Setting	Value
Route	☒
Route Settings	Route Settings
Trim Trunks	☒
Gate Layer	Metal2
Gate Width	0.08
Gate Offset	0.07
Source Layer	Metal2
Source Width	0.08
Source Offset	0.07
Drain Layer	Metal2
Drain Width	0.08
Drain Offset	0.07
Bulk Layer	Metal2
Bulk Width	0.08
Bulk Offset	0.07
Channel0 Device Spacing	0.14
Channel1 Device Spacing	0.14

- **Trim Trunks:** This check box is selected by default. It means that while routing, both the ends of a trunk will be trimmed to the first twig on that trunk. If you deselect this check box, trunk trimming gets disabled and the length of the trunk will be the same as the width of the modgen.
- **Gate Layer:** Select a gate layer on which a horizontal trunk should be created.
- **Gate Width:** Enter a value that defines the gate width of the horizontal trunk when it is created.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

- ☐ *Gate Offset:* Enter the required gate offset (spacing) between horizontal trunks in the same channel.
 - ☐ *Source Layer:* Select a source layer on which a horizontal trunk should be created.
 - ☐ *Source Width:* Enter a value that defines the source width of the horizontal trunk when it is created.
 - ☐ *Source Offset:* Enter the required source offset (spacing) between horizontal trunks in the same channel.
 - ☐ *Drain Layer:* Select a drain layer on which a horizontal trunk should be created.
 - ☐ *Drain Width:* Enter a value that defines the drain width of the horizontal trunk when it is created.
 - ☐ *Drain Offset:* Enter the required drain offset (spacing) between horizontal trunks in the same channel.
 - ☐ *Bulk Layer:* Select a bulk layer on which a horizontal trunk should be created.
 - ☐ *Bulk Width:* Enter a value that defines the bulk width of the horizontal trunk when it is created.
 - ☐ *Bulk Offset:* Enter the required bulk offset (spacing) between horizontal trunks in the same channel.
 - ☐ *Channel0 Device Spacing:* Specify the spacing between the last (bottom) trunk in the channel and the devices immediately below it.
 - ☐ *Channel1 Device Spacing:* Specify the spacing between the first (top) trunk in the channel and the devices immediately above it.
-

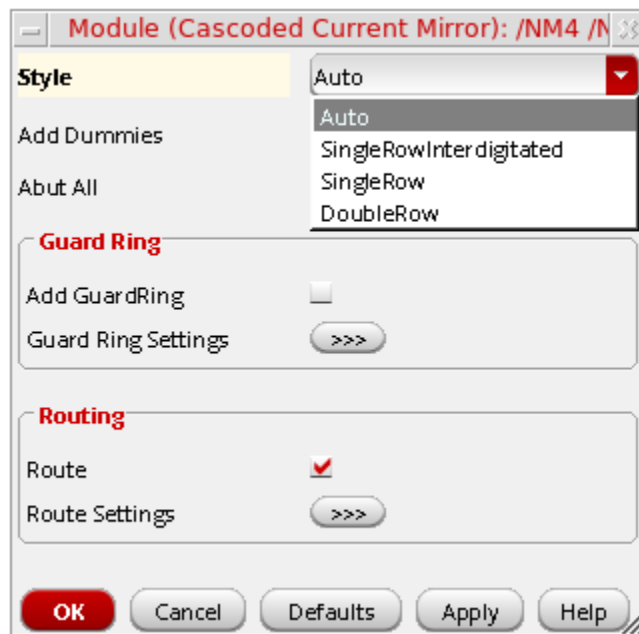
Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

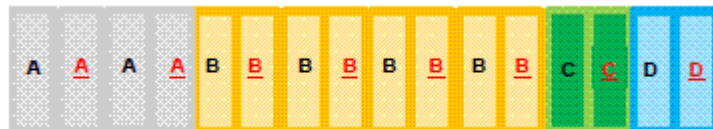
MOS Cascoded Current Mirrors

It will generate a pattern similar to the MOS Current Mirror with an alternate between current mirror device and cascoded device. Select the required pattern from the *Style* drop-down list box in the Module (Cascoded Current Mirror) form.

As shown in the figure below, the following pattern options are available in the *Style* drop-down list box:



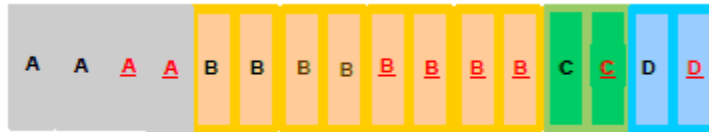
- *Style (Auto)*: Automatically picks between *SingleRowInterdigitated*, *SingleRow*, or *DoubleRow* styles based on the aspect ratio of the different possible row counts.
- *Style (SingleRowInterdigitated)*: Places all the devices in a single row. The devices are processed in a schematic order. The mfactored device and its cascode are added in an interdigitated alternating pattern.



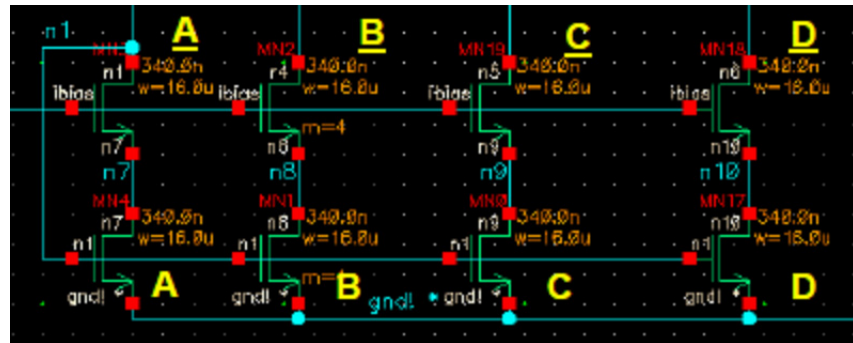
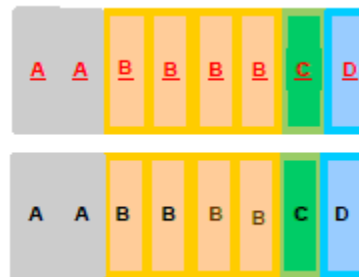
Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

- **Style (SingleRow):** Places all the devices in a single row. The devices are processed in a schematic order. The mfactored device and its cascode are added in an alternating pattern.



- **Style (DoubleRow):** Creates a two row pattern, which orders the devices as they are in the schematic.



- **Add Dummies:** Sets the standard dummies option.
- **Guard Ring:** Controls whether a guard ring should be added. Use the *Guard Ring Settings* button to view and set the standard guard ring options shown in the figure below. The standard guard ring options available in the *Guard Ring* group box are the same as discussed in the [Adding a Guard Ring Around Generated Modgen](#) section of Chapter 1, “The Constraint Manager Assistant.”

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

- **Routing:** Controls whether pin to trunk routing should be enabled. Use the *Route Settings* button to view and set the standard routing options. For description of the available standard routing options, refer to [Routing](#) explained with the [MOS Cascode](#) RAP finder description.

This finder group contains at least four devices registered as "fet". For more information, refer to the [MOS Cascoded Current Mirror](#) section of the Structure finder.

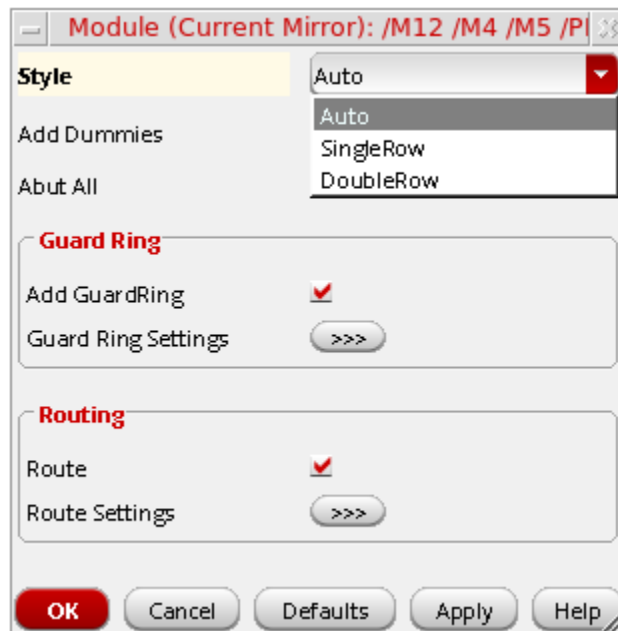
MOS Cascoded Current Mirrors2

It has a same pattern as [MOS Current Mirror](#) with an alternate between current mirror device and cascoded device.

This finder group contains at least four devices registered as "fet". For more information, refer to the [MOS Cascoded Current Mirror2](#) section of the Structure finder.

MOS Current Mirror

This finder controls the modgen generation. As shown in the figure below, it has the following options in the *Style* drop-down list box:



- **Style (Auto):** Automatically picks the best number of rows based on the aspect ratio of the different possible row counts.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

- **Style (SingleRow):** Sorts the devices by mfactor based on the number of fingers. It places the largest mfactored devices at the two ends of a modgen and the smaller mfactored devices in between them.



- **Style (DoubleRow):** It is same as the *SingleRow* option, but it places the mfactored devices in two rows.



- **Add Dummies:** Sets the standard dummies option.
- **Guard Ring:** Controls whether a guard ring should be added. Use the *Guard Ring Settings* button to view and set the standard guard ring options shown in the figure below. The standard guard ring options available in the *Guard Ring* group box are the same as discussed in the Adding a Guard Ring Around Generated Modgen section of Chapter 1, "The Constraint Manager Assistant."
- **Routing:** Controls whether pin to trunk routing should be enabled. Use the *Route Settings* button to view and set the standard routing options. For description of the available standard routing options, refer to Routing explained with the MOS Cascode RAP finder description.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

This finder group contains at least two devices registered as "fet". For more information, refer to the [MOS Current Mirror](#) section of the Structure finder.

If the *Add Dummies* check box is checked on the Constraint Generator Dialog, dummy devices will be added to either end of each row of the generated modgen. Alternatively, you can set the [currentMirrorIncludeDummies](#) environment variable to `t`.

The first device in the matched structure acts as the reference device for the dummy device. The dummy device will have the same `libName`, `cellName`, `width`, `length`, `fingerCount`, and `fingerWidth` as the reference device.

The net to be used to connect the dummy device terminals are determined from the bulk connectivity of the devices in the matched structure, or the power/ground nets connected to the devices in the matched structure if the devices have no bulk.

MOS Differential Pair

Lists the regular differential pair structures. If the [diffPairRequireSourceNotSupply](#) environment variable is not set, all differential pair structures will be recognized and listed. However, if you set this environment variable to `t`, the differential pair structures will be listed only if the source connection of the structure is not connected to a supply net.

It will generate patterns illustrated below based on the option you select from the *Style* drop-down list box in the Module (Diff Pair) form.

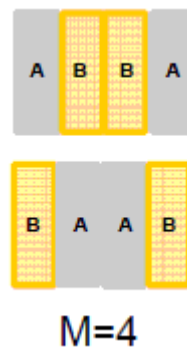
Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

As shown in the figure below, the following options are available in the *Style* drop-down list box:



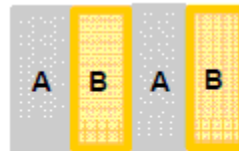
- *Style (Auto)*: Automatically picks between row1 and row2symmetric styles based on the device mfactors.



Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

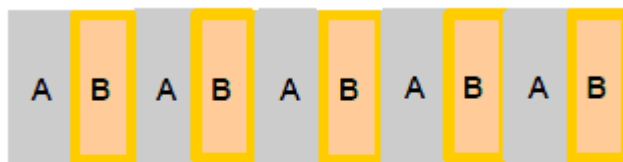
- *Style (row1)*: Simple one row alternating pattern.



M=2



M=3

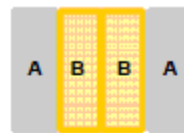


M=5

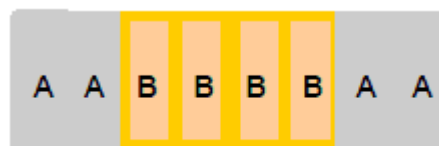
Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

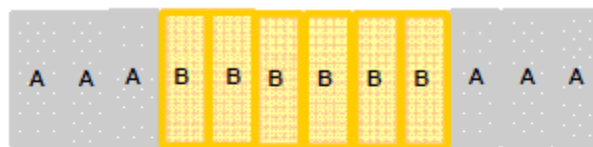
- *Style (row2symmetric)*: If the mfactor is a multiple of 4 then a 2 row cross quad pattern will be used.



M=4



M=8

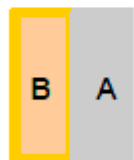


M=12

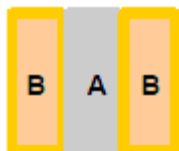
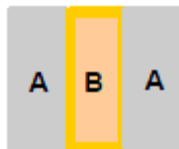
Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

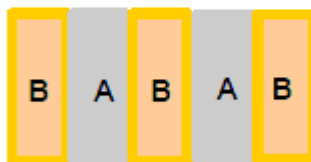
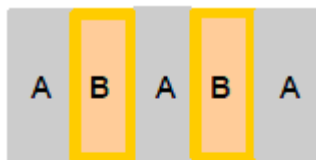
If the mfactor is not a multiple of 4 and Auto/row2symmetric style is selected, then a 2 row alternating pattern (ABAB...) will be used.



M=2



M=3



M=5

- *Add Dummies*: Sets the standard dummies options.
- *Guard Ring*: Controls whether a guard ring should be added. Use the *Guard Ring Settings* button to view and set the standard guard ring options shown in the figure below. The standard guard ring options available in the *Guard Ring* group box are the same as discussed in the [Adding a Guard Ring Around Generated Modgen](#) section of Chapter 1, “The Constraint Manager Assistant.”
- *Routing*: Controls whether pin to trunk routing should be enabled. Use the *Route Settings* button to view and set the standard routing options. For description of the available standard routing options, refer to [Routing](#) explained with the [MOS Cascode](#) RAP finder description.

For more information, refer to the [MOS Differential Pair](#) section of the Structure finder.

MOS Differential Pair - Cross Coupled

Lists the cross-coupled differential pair structures. Such structures are similar to the regular differential pair structures, but additionally the gate terminal of each transistor connects to the drain terminal of the other transistor (the two sources are connected as with any differential pair structure).

If the [crossCoupleDiffPairRequireSourceNotSupply](#) environment variable is not set, all cross-coupled differential pair structures will be recognized and listed. However, if you set this environment variable to `t`, which is the default, the cross-coupled differential pair structures will be listed only if the source connection of the structure is not connected to a supply net.

This finder generates same patterns as the [MOS Differential Pair](#) finder based on the option you select from the *Style* drop-down list box in the Module (Diff Pair - Cross Coupled) form. The *Style* drop-down list provides the same options as the *MOS Differential Pair* finder.

For more information, refer to the [MOS Differential Pair](#) section of the Structure finder.

Virtuoso Unified Custom Constraints User Guide

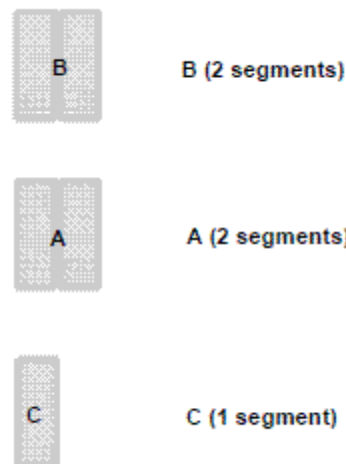
The Circuit Prospector Assistant

Passive Arrays

Groups together series/parallel connected resistors and capacitors. The Constraint Generator generates a modgen for each group.

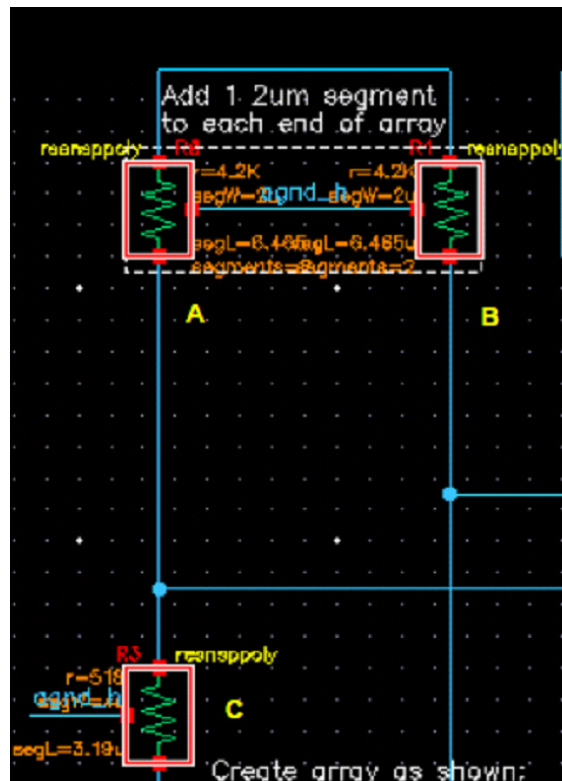


- **Style (auto):** Automatically picks the best pattern based on the mfactor to give the squarest modgen. For example, if there are four devices, you will have a 2x2 pattern (a gap might be there if the number of devices is not a power of 2).
- **Style (Row1..RowN):** Allows you to pick num rows, where N is the mfactor.



Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant



- **Add Dummies:** Standard dummies option, as Current Mirror.
- **Guard Ring:** Controls whether a guard ring should be added. Use the *Guard Ring Settings* button to view and set the standard guard ring options shown in the figure below. The standard guard ring options available in the *Guard Ring* group box are the same as discussed in the Adding a Guard Ring Around Generated Modgen section of Chapter 1, "The Constraint Manager Assistant."
- **Routing:** Controls whether pin to trunk routing should be enabled. Use the *Route Settings* button to view and set the standard routing options. For description of the available standard routing options, refer to Routing explained with the MOS Cascode RAP finder description.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

Active Same Cell large mfactor Defines large mfactor device as `mfactor*fingerCount > 5` that can be modified by you.

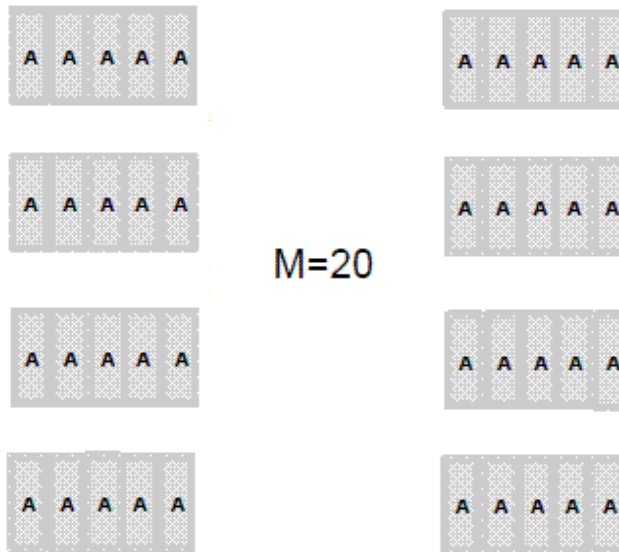


- *Style (Auto)*: Picks the best pattern automatically based on the `mfactor` to give the squarest modgen with no gaps.
 - *Style (row1 .. rowN)*: Allows you to pick number of rows. Here, N is the `mfactor`.
-

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

- **Guard Ring:** Controls whether a guard ring should be added. Use the *Guard Ring Settings* button to view and set the standard guard ring options shown in the figure below. The standard guard ring options available in the *Guard Ring* group box are the same as discussed in the [Adding a Guard Ring Around Generated Modgen](#) section of Chapter 1, “The Constraint Manager Assistant.”



MOS/Bipolar

Passive

- **Routing:** Controls whether pin to trunk routing should be enabled. Use the *Route Settings* button to view and set the standard routing options. For description of the available standard routing options, refer to [Routing](#) explained with the [MOS Cascode RAP](#) finder description.
-

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

Symmetric Instance Pairs - By Connectivity

Groups pairs of instances that are determined to be symmetric based on their connectivity. This finder searches for only symmetric instance pairs.

The *Symmetric Instance Pairs - By Connectivity* and Symmetric Net Pairs iterators initially look for symmetrical triggering pairs in the current schematic. From these pairs, they will propagate symmetries along the nets and devices that are connected to the triggering pairs. There are three types of symmetrical triggering pairs:

1. A pair of instances or nets with an existing symmetry constraint.
 2. A differential pair made up of `fet` and `bjt` devices (see below on how to define both of these categories).
 3. A pair of instances with Active Same Cell Name and Same Size `fet` and `bjt` devices, with mirrored orientation, aligned on the same `Y` coordinate.
-

Note:

- The symmetries are propagated through nets using terminal names defined for each active device.
- Symmetries are also transmitted to, and propagated through, passive devices.
- To be a symmetrical pair, both instances must have the same cell name and size.
- Symmetries for instances, or nets, are converted to self-symmetries when there is only one member on the path for symmetry propagation.
- `fets`, `bjts` and passive must be registered for each PDK for the *Circuit Prospector* to identify symmetries.

To see the examples on `fet` and `bjt` devices, refer to the Example of `fet` and `bjt` devices defined: section in the Structure finder.

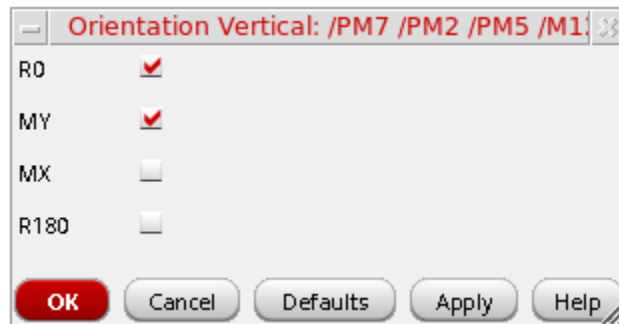
Capacitor Cluster

Finds all capacitors to generate cluster constraints for them. This is necessary to prevent capacitors being placed on top of other devices.

Virtuoso Unified Custom Constraints User Guide

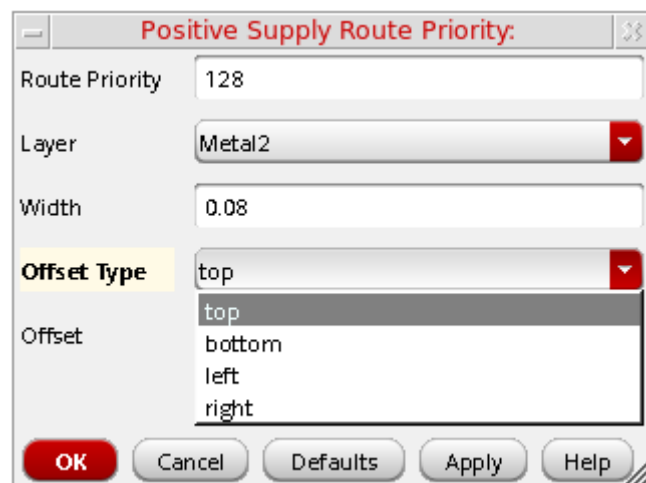
The Circuit Prospector Assistant

Vertical Orientation Sets the specified vertical orientation (R0 /MY/MX/R180) constraints on all selected `fets` and `bjts` devices.



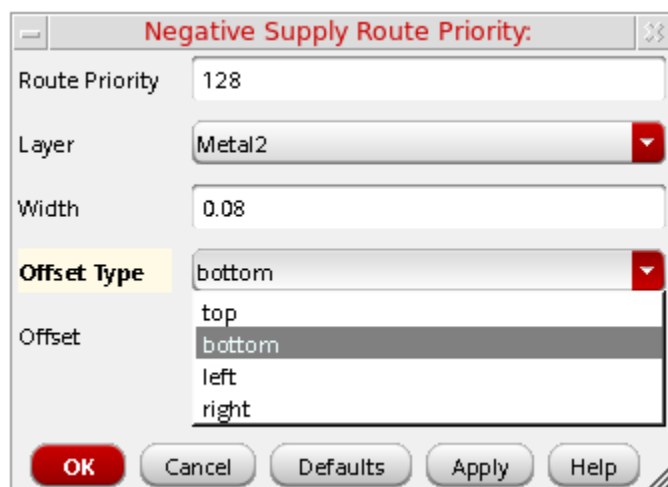
Positive/Negative Supply Groups the positive or negative supply nets based on the type of finder chosen. Positive and negative supply nets are identified by matching their names against the nets registered as power/ground through `ciRegisterNet()` or through their `sigTypes`.

Using these two RAP finders, you can generate high priority constraints for the current cellview or each sub-cell. When you click the required constraint generator, one of the following forms is displayed depending on the chosen finder.



Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant



For each power or ground net, the following constraints are created:

- A Net Priority constraint
- A Rail constraint with the chosen layer, width, side, and offset



<i>Nets (Symmetry By Connectivity)</i>	This option enables you to create symmetry constraints on symmetrical net/inst pairs or self-symmetric nets/insts.
<i>Top Pins (Alignment)</i>	Creates an alignment constraint on all schematic pins that are placed on the top of the schematic.
<i>Bottom Pins (Alignment)</i>	Creates an alignment constraint on all schematic pins that are placed on the bottom of the schematic.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

<i>Left Pins (Alignment)</i>	Creates an alignment constraint on all schematic pins that are placed on the left of the schematic.
<i>Right Pins (Alignment)</i>	Creates an alignment constraint on all schematic pins that are placed on the right of the schematic.
<i>Enforce Precedence</i>	The <i>Enforce Precedence</i> finder: ■ Enforces a single symmetry line. ■ Moves symmetry constraint from two symmetrical devices onto the modgens they are contained within. ■ Moves self-symmetry from a device to the modgen it is contained within if it is the only device in that modgen. ■ Deletes cluster members if the members are within modgens. ■ The alignment constraint on symmetrical pins is removed to ensure that the pin symmetry takes precedence.
<i>ALL</i>	Executes all finders of the current category sequentially in the order of their listing in the <i>Search for</i> list and displays the corresponding results in the browser.

Structures

Structures can be [Devices](#), [Nets](#), or [Pins](#).

Note: Structure finders do not work across hierarchical boundaries, as most structures only exist within the same cell.

All [Finders](#) related to structures are detailed below. Finder order is important and must follow the order given.

Structure finders can use “*insts*”, “*nets*”, and “*pins*” (lists of database objects found by structures finders) as variables for matched expressions. For example, for a *MOS Differential Pair* you could use:

```
car(insts)->cellName == cadr(insts)->cellName
```

Note: Although the structures listed below only act on [Devices](#), it does not limit the scope of what you can create.

For information on adding new structures, see [Capture Circuit Structure Form](#).

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

For information on mapping structures, see [ciMapTerm](#), [ciPrintMappedParams](#), and [ciGetDeviceTermName](#).

Structures Finder	Description
<i>MOS Cascode</i>	Finds cascoded MOS transistor structures. This finder group contains at least two devices registered as "fet". G0 => D0 Gi => Gi+1 eg: G0->G1->G2->etc. Si => Di+1 eg : S0->D1 ; S1->D2 ; etc.
<i>MOS Current Mirror</i>	Groups devices that form a MOS current mirror structure. This finder group contains at least two devices registered as "fet". S1 = S2 (=S3, etc.) B1 = B2 (=B3, etc.) G1 = G2 (=G3, etc.) D1 != D2 (!=D3, etc.) D1 = G1 XOR D2 = G1 (XOR D3 = G1, etc.) Note: Use the currentMirrorIncludeDummies environment variable to control the inclusion of dummies in Current Mirror structures.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

MOS Cascoded Current Mirror

Groups devices that form a cascoded MOS current mirror structure.

This finder group contains at least four devices registered as "fet".

```
G1a = G2a ( = G3a, etc.)
G1b = G2b ( = G3b, etc.)
B1a = B2a ( = B3a, etc.)
B1b = B2b ( = B3b, etc.)
D1a != D2a ( != D3a, etc.)
D1b != D2b ( != D3b, etc.)
D1a = S1b
D2a = S2b
(D3a = S3b, etc.)
G1a != G1b
S1a != S1b
D1a != D1b
D1b = G1a XOR D2b = G1a (XOR D3b = G1a, etc.).
```

MOS Cascoded Current Mirror2

Groups devices that form a cascoded MOS current mirror structure with at least one of the leading pairs being diode connected.

This finder group contains at least four devices registered as "fet".

```
G1a = G2a ( = G3a, etc.)
G1b = G2b ( = G3b, etc.)
B1a = B2a ( = B3a, etc.)
B1b = B2b ( = B3b, etc.)
D1a != D2a ( != D3a, etc.)
D1b != D2b ( != D3b, etc.)
D1a = S1b
D2a = S2b
(D3a = S3b, etc.)
G1a != G1b
S1a != S1b
D1a != D1b
(D1a = G1a XOR D2a = G1a (XOR D3a = G1a, etc.)) (D1b = G1b
XOR D2b = G1b (XOR D3b = G1b, etc.))
```

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

MOS Transmission Gate

Groups devices that form a MOS transmission gate structure.

This finder group contains two devices:

- Transistor M1 that must be registered as an "nfet".
- Transistor M2 that must be registered as an "pfet".

The names of their terminals must be registered as "drain", "gate", and "source" respectively.

Their respective drain, source, and gate terminals must be connected as follows:

D1 == D2

S1 == S2

G1 != G2

MOS Differential Pair

Groups devices that form a MOS differential pair structure.

This finder is based on terminals and is used for matching all electrical parameters (fingers width, fingers, length, multiplier). Cell name must match.

S1 = S2 and not connected to power net.

B1 = B2

G1 != G2

D1 != D2

G1 != D1

G1 != S1

D1 != S1

G2 != D2

There is an equivalent mapping for bjt devices:

("S" -> "E", "B" -> null, "G" -> "B", "D" -> "C")

Structure finders can use "insts", "nets", and "pins" (lists of database objects found by structures finders) as variables for matched expressions. For example, for a *MOS Differential Pair*, you could use:

`car(insts)->cellName == cadr(insts)->cellName`

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

MOS Differential Pair - Cross Coupled

Groups devices that form a MOS cross-coupled differential pair structure.

This finder group contains two devices registered as "fet".

D1 = G2

D2 = G1

S1 = S2

B1 = B2

D1 != G1

D1 != S1

G1 != S1

D2 != G2

Note: There is an equivalent mapping for bjt devices:
("S" -> "E", "B" -> null, "G" -> "B", "D" -> "C")

MOS Parallel

Groups devices that form a MOS parallel structure.

This finder group contains at least two devices registered as "fet".

S1 = S2

B1 = B2

G1 = G2

D1 = D2

There is an equivalent mapping for bjt devices:
("S" -> "E", "B" -> null, "G" -> "B", "D" -> "C")

MOS Active Load

Groups devices that form a MOS active load structure.

This finder group contains at least two devices registered as "fet".

D1 = one of differential pair D

D2 = the other differential pair

B1 = B2

G1 = G2

S1 = S2 (and be connected to a power net)

D1 = G1 XOR D2 = G1

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

MOS Inverter

Groups devices that form a MOS inverter structure.

This finder group contains two devices.

- Transistor M1 that must be registered as an "nfet".
- Transistor M2 that must be registered as an "pfet".

The names of their terminals must be registered as "drain", "gate", and "source" .

Their respective drain, source and gate terminals must be connected as follows:

G1 = G2

D1 = D2

S1 != S2

S1 connected to power supply

S2 connected to a different power supply

Symmetric Instance Pairs - By Pre-selection

Groups pairs of instances that are geometrically symmetrical about the center of the current selected set.

Note: Instances in geometrically symmetrical regions of the schematic will need to be selected prior to running this finder.

Symmetric Instance Pairs - By Connectivity

Groups pairs of instances that are determined to be symmetric based on their connectivity. This finder searches for only symmetric instance pairs.

The *Symmetric Instance Pairs - By Connectivity* and Symmetric Net Pairs iterators initially look for symmetrical triggering pairs in the current schematic. From these pairs, they will propagate symmetries along the nets and devices that are connected to the triggering pairs. There are three types of symmetrical triggering pairs:

1. A pair of instances or nets with an existing symmetry constraint.
 2. A differential pair made up of `fet` and `bjt` devices (see below on how to define both of these categories).
 3. A pair of instances with Active Same Cell Name and Same Size `fet` and `bjt` devices, with mirrored orientation, aligned on the same Y coordinate.
-

Note:

- The symmetries are propagated through nets using terminal names defined for each active device.
- Symmetries are also transmitted to, and propagated through, passive devices.
- To be a symmetrical pair, both instances must have the same cell name and size.
- Symmetries for instances, or nets, are converted to self-symmetries when there is only one member on the path for symmetry propagation.
- `fets`, `bjts` and `passive` must be registered for each PDK for the *Circuit Prospector* to identify symmetries.

Example of `fet` and `bjt` devices defined:

```
ciRegisterDevice("fet"  
  '( (nil "nmos" nil)  
    (nil "pmos" nil)  
    (nil "nmos1v" nil)  
    ...  
  ) )  
  
ciRegisterDevice("bjt"  
  '( (nil "npn" nil)  
    (nil "pnp" nil)  
    (nil "vpnp2" nil)  
    ...  
  ) )
```

Terminal names must be registered in the respective list of terminal place-holder names. The first one that is found from the list is used by the iterator.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

- “drain”, “gate”, “source”, and “bulk” are used for fet devices.
- “collector”, “base”, “emitter”, and “bulk” are used for bjt devices.

For example:

```
ciMapTerm(("drain" '("d" "drain" "DRAIN" "Drain"))
ciMapTerm("gate" '("g" "gate" "GATE" "Gate"))
ciMapTerm("source" '("s" "source" "SOURCE"
"Source")) ciMapTerm("bulk" '("B" "S" "BULK" "well"
"SUB")) ciMapTerm("collector" '("c" "C" "collector"))
ciMapTerm("base" '("b" "B" "base" "BASE" "Base"))
ciMapTerm("emitter" '("e" "E" "emitter"))
```

See also [Nets](#)

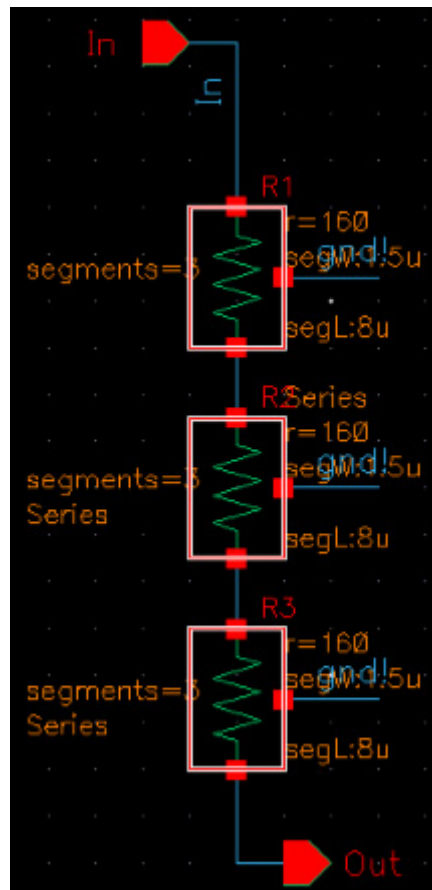
Self Symmetric Instances - By Connectivity

Groups the instances that are determined to be self-symmetric based on their connectivity.

Placement Path

Identifies current paths in the active schematic cellview and creates a Placement Path constraint in the schematic Constraint Manager.

Series Resistor Array Groups arrays of sequentially-arranged resistors that form the longest serial chain between two nets. The series chain terminates at a branch that connects to multiple resistors. Branches that connect to devices other than resistors are ignored.



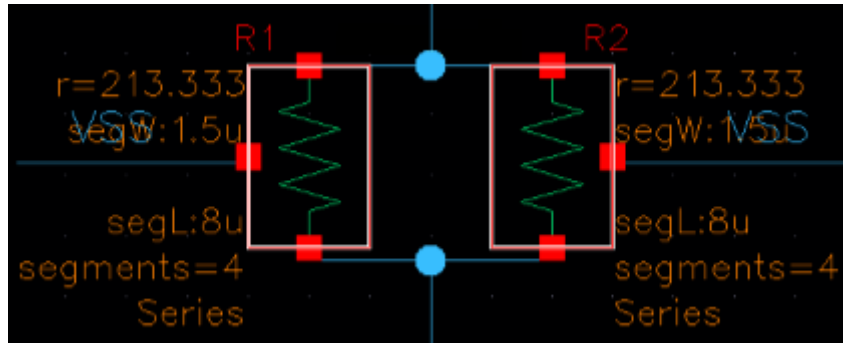
Using the *Series Resistor Array* finder, you can generate modgen constraints for resistor arrays for the current cellview or each sub-cell. When you click the required constraint generator, a form is displayed for specifying the required criteria for creating a modgen template for the selected series resistor arrays. For detailed information about the components of the displayed form, see [Creating Modgen Template for Resistor Arrays](#).

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

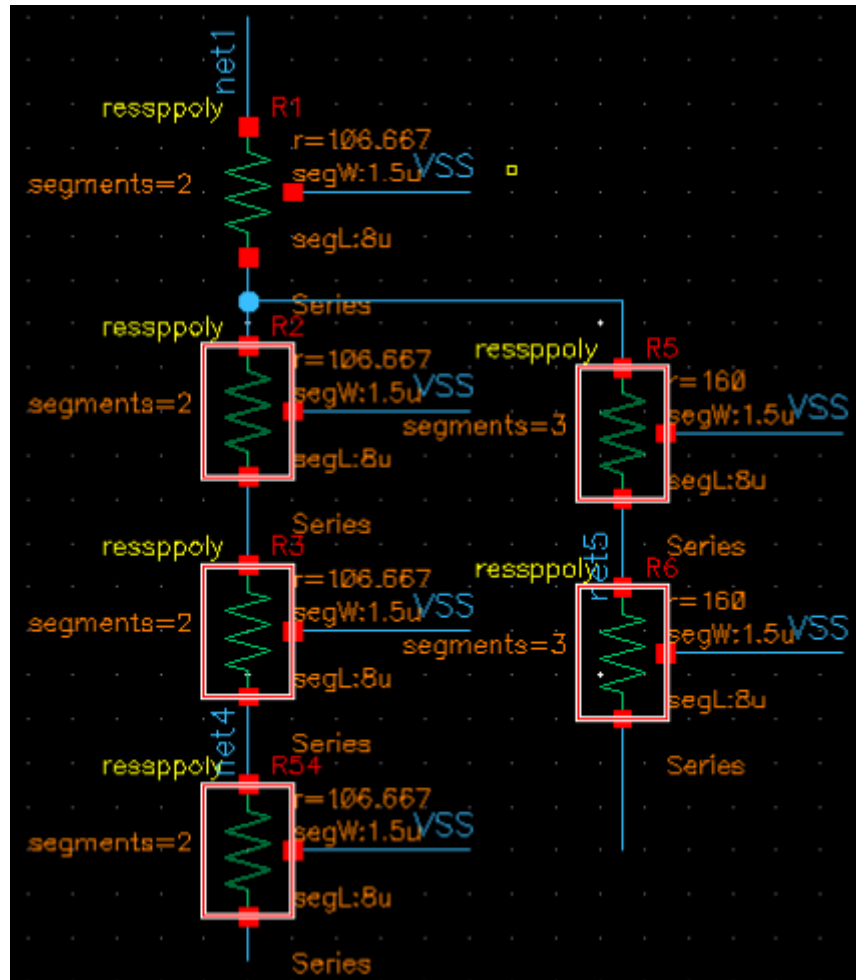
Parallel Net Resistor Array

Groups arrays of parallel-arranged resistors that start and end on the same net.



Using the *Parallel Net Resistor Array* finder, you can generate modgen constraints for resistor arrays for the current cellview or each sub-cell. When you click the required constraint generator, a form is displayed for specifying the required criteria for creating a modgen template for the selected series resistor arrays. For detailed information about the components of the displayed form, see [Creating Modgen Template for Resistor Arrays](#).

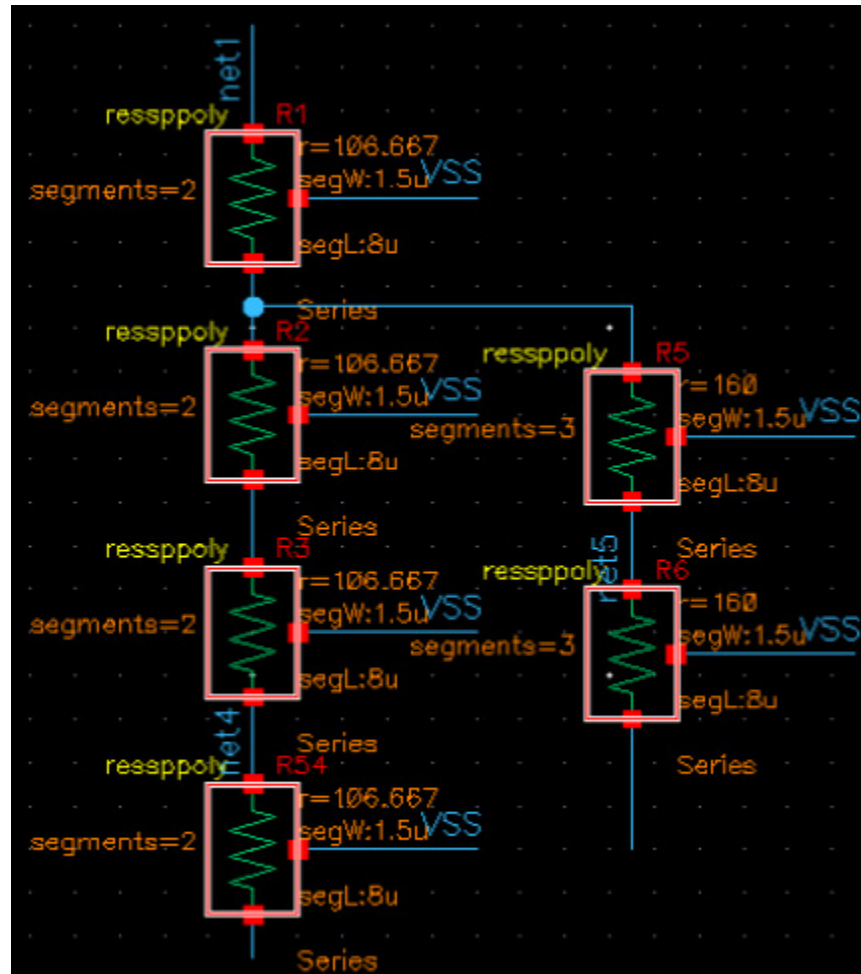
Parallel Resistor Array (Length) Groups arrays of parallel-arranged resistors that are all of the same length when their m-factor and iteration are expanded.



Using the *Parallel Resistor Array (Length)* finder, you can generate modgen constraints for resistor arrays for the current cellview or each sub-cell. When you click the required constraint generator, a form is displayed for specifying the required criteria for creating a modgen template for the selected series resistor arrays. For detailed information about the components of the displayed form, see [Creating Modgen Template for Resistor Arrays](#).

Note: The constraint generator for the *Parallel Net Resistor Array* and *Parallel Resistor Array (Length)* finders is the same. Therefore, the form displayed for both is also the same.

<i>Block Resistor Array</i>	Groups all resistors in the design to form a large block. The block can include any number of resistor chains.



Using the *Block Resistor Array* finder, you can generate modgen constraints for resistor arrays for the current cellview or each sub-cell. When you click the required constraint generator, a form is displayed for specifying the required criteria for creating a modgen template for the selected series resistor arrays. For detailed information about the components of the displayed form, see [Creating Modgen Template for Resistor Arrays](#).

<i>ALL</i>	Finds all of the listed structures sequentially in the order of their listing in the <i>Search for</i> list.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

Devices

Examples of devices include:

- fets (field effect transistors)
- mosfets (metal oxide semiconductor field effect transistors)
- bjts (bipolar junction transistors)
- hbts (heterojunction bipolar transistors)
- soifets (silicon on insulator field effect transistors)
- hemts (high electron mobility transistors)

Most Finders in this category group devices by cell type, the exception being *Active Same Well*, where the benefits of debugging an incorrect body connection outweighs the false positives avoided by adding a same cell name restriction.

All Finders related to devices are discussed below:

Devices Finder	Description
<i>Active Common Gate</i>	Group active devices with same cell name. The result here will comprise of one or more devices registered as "fet" or "bjt", whatever the cell names, and for the following connectivity: G1 (=G2, etc) (common gate for "fet" devices) B1 (=B2, etc) (common base for "bjt" devices)
<i>Active Same Cell</i>	Identifies all instances with the same cell name. You will be displayed all device groups that contain the same cell name, for example all pmos3 may be outlined in red and all pmos4 outlined in blue, and so on. You can then cycle through the groups and apply constraints, after which devices will be haloed (see <u>Creating User-Defined Constraints</u> in the <i>Virtuoso Unified Custom Constraints Configuration Guide</i>). Associated Constraint: Group 1: Orientation Orientations= R0, R180, MX, MY

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

<i>Active Same Cell and Common Gate</i>	Groups active devices with same cell name connected with a common gate.
<i>Active Same Cell and Common Bulk</i>	Group active devices with same cell name and common bulk.
<i>Active Same Cell, Common Gate, and Bulk</i>	Group active devices with same cell name with common gate and bulk. The result here will comprise of one or more devices registered as "fet" or "bjt". The cell names must be the same and the connectivity must be as follows: G1 (=G2, etc) (common gate for "fet" devices) B1 (=B2, etc) (common base for "bjt" devices) BU1 (=BU2, etc) (common bulk)
<i>Active Same Cell and Finger Width</i>	Group active devices with same cell name and finger width to set a Matched Parameters constraint.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

Active Same Cell and Size

Identifies all instances that match the editable parameters, for example instance parameters in the CDF.

All device groups that match in editable parameters will be displayed, for example all `pmos3` outlined in red, all `pmos4` outlined in blue, and so on. You can then cycle through the groups and apply constraints, after which, weak outlining will be applied for found devices, and haloing for constrained devices (see [Creating User-Defined Constraints](#) in the *Virtuoso Unified Custom Constraints Configuration Guide*).

Active devices will be grouped by cell name and size. For example, if device M1 is an `nmos` and has: `w=2u`, `l=130nm`, `m=1`, it will not be grouped with M2 which is a `pmos` of the same size.

The matching expression for *Active Same Cell and Size* is:

```
if(ciIsDevice(device "fet") then
list(ciGetParamName(device "length")
ciGetParamName(device "width")) else nil)
```

Note: The *Active Same Cell and Size* finder is particular useful for electrical constraints.

Associated Constraint:

Group 1:

MatchedOrientation Orientations= R0, R180, MX, MY

Matched Parameters:

MatchedSubset=nil

Passive Same Cell Name

Identifies all instances with the same cell name.

You will be displayed all device groups that contain the same cell name, for example all `pmos3` may be outlined in red and all `pmos4` outlined in blue, and so on. You can then cycle through the groups and apply constraints, after which devices will be haloed (see [Creating User-Defined Constraints](#) in the *Virtuoso Unified Custom Constraints Configuration Guide*).

Associated Constraint:

Group 1: Orientation Orientations= R0, R180, MX, MY

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

Passive Same Cell Name and Size

Identifies all instances that match editable parameters, for example instance parameters in the CDF.

All device groups that match in the editable parameters will be displayed, for example all `pmos3` outlined in red, all `pmos4` outlined in blue, and so on. You can then cycle through the groups and apply constraints, after which, weak outlining will be applied for found devices, and haloing for constrained devices (see [Creating User-Defined Constraints](#) in the *Virtuoso Unified Custom Constraints Configuration Guide*).

The matched expression matches `length`, `width`, `resValue`, `indValue`, and `capValue`. For example:

```
if(ciIsDevice(device "passive") then
list(ciGetParamName(device "length")
ciGetParamName(device "width") ciGetParamName(device
"resValue") ciGetParamName(device indValue")
ciGetParam(device "capValue")) else nil)
```

Associated Constraint:

Group 1: MatchedOrientation Orientations= R0, R180, MX, MY

Matched Parameters:

MatchedSubset=nil.

ALL

Executes all finders of the current category sequentially in the order of their listing in the *Search for* list and displays the corresponding results in the browser.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

Nets

Examples of nets are power, ground, and symmetry. All [Finders](#) related to nets are detailed below:

Net Finder	Description
<i>Supply Nets</i>	Represents all power/ground nets in the current cellview. Supply nets are recognized by name, and the names can be set by calling <code>ciRegisterNet</code> with a list of supply net names, for example <code>ciRegisterNet("supply" list("agnd" "agnd!" "avdd" "avdd!"))</code>. The supply net finder uses the <code>ciIsNet(net)</code> function to check the net name against the list of supply net names (all names are converted to lower case).
<i>Non-Supply Nets</i>	Represents all nets in the current cellview that are <i>not</i> supply nets. For example, the finder expression is: <pre>!ciIsNet(net "supply")</pre>
<i>Symmetric Net Pairs</i>	Groups pairs of nets that are determined to be symmetric based on their connectivity. For full description of operation see Symmetric Instance Pairs - By Connectivity.
<i>Bus Bundle</i>	Creates a bus for all signals of a bus net.
<i>Registered (sigType)</i>	Creates a sigType property for each registered net.
<i>Same sigType (None)</i>	Group all nets with the same sigType property value.
<i>ALL</i>	Finds all supply and non-supply nets, and all symmetric net pairs sequentially in the order of their listing in the <i>Search for</i> list.

Pins

Pins are members of symmetry, alignment, and distance. All [Finders](#) related to pins are detailed below:

Pins Finders	Description
<i>Pins (Symmetry By Connectivity)</i>	Group pairs of pins that are determined to be symmetric based on connectivity.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

<i>Top Pins (Alignment)</i>	Group all top pins together.
<i>Bottom Pins (Alignment)</i>	Group all bottom pins together.
<i>Left Pins (Alignment)</i>	Group all left pins together.
<i>Right Pins (Alignment)</i>	Group all right pins together.
<i>Supply Pins (Alignment)</i>	Group all left pins together.
<i>Input Pins (Alignment)</i>	Group all input pins together.
<i>Output Pins (Alignment)</i>	Group all output pins together.
<i>ALL</i>	Finds all pins sequentially in the order of their listing in the <i>Search for</i> list.

Inst Terms

All [Finders](#) related to Inst Terms are detailed below:

Inst Term Finders	Description
<i>InstTerms by Instance</i>	Groups all instTerms for the same instance together.
<i>InstTerms by Name</i>	Group all instTerms by their name.
<i>InstTerms by Net</i>	Group all instTerms connected to the same net.
<i>ALL</i>	Finds all inst terms sequentially in the order of their listing in the <i>Search for</i> list.

Other Pre-defined Categories (Disabled by Default)

This section covers details about the following pre-defined categories that are disabled by default, but can be enabled using the [ciEnableAssistant](#) SKILL function:

- [Properties](#)
- [Electrical Constraints](#)
- [Placement Constraints](#)
- [Routing Constraints](#)
- [Clusters](#)

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

- [Matched Parameters](#)
- [Modgens](#)
- [Orientations](#)
- [Symmetries](#)
- [Analog Device Structures](#)

Properties

All [Finders](#) related to Properties are detailed below:

Properties Finders	Description
<i>Registered Nets</i> (<i>sigType</i>)	Creates a <code>sigType</code> for each registered net.
<i>Same sigType Nets</i> (<i>none</i>)	Group all nets with same <code>sigType</code> property value.
<i>fets</i> (<i>variantInfo</i>)	Create a <code>variantInfo</code> property on FET devices with <code>w > 5u</code> with pairs of <code>fingerCount</code> and <code>fingerWidth</code> .
<i>Same variantInfo</i> <i>Devices</i>	Group all devices with the same <code>variantInfo</code> property value.
<i>pfets</i> (<i>placerControlledWell</i>)	Create a <code>placerControlledWell</code> property for each PMOS device.
<i>Same</i> <i>placerControlledWell</i> <i>Devices</i>	Group all devices with the same <code>placerControlled</code> property value. Note: The <code>placerControlledWell</code> property is only created if a layer with the function "nwell" is defined in the <code>layerRules</code> section of the technology file. The corresponding layer name is used to set the value of the property.
<i>ALL</i>	Finds all properties sequentially in the order of their listing in the <i>Search for</i> list.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

Electrical Constraints

All Finders related to Electrical Constraints are detailed below:

Electrical Constraints Finders	Description
<i>Symmetric Pairs (Correlation)</i>	Group instances by symmetrical pairs according to connectivity to set a correlation constraint.
<i>Same Cell, Size, Bulk Devices (Matched Parameters)</i>	Group devices with same cell name, bulk, connection and size (w, l, r, c)
<i>ALL</i>	Finds all electrical constraints sequentially in the order of their listing in the <i>Search for</i> list.

Placement Constraints

All Finders related to Placement Constraints are detailed below:

Placement Finders	Description
<i>Common Bulk (Cluster)</i>	Group devices with same cell names and a common bulk.
<i>Digital Cells (Cluster)</i>	Group all standard cells together. To be captured, all standard cells must be registered with the function: <pre>ciRegisterDevice("standardCell" list(list("libName" "cellName" "viewName") list(LCV2) ...)</pre>
<i>Same Cell, Size, Bulk Devices (Modgen)</i>	Group devices with same cell name, bulk connection and size (w, l, r, c).
<i>Clusters (Cluster Boundary)</i>	Create a Cluster Boundary for any cluster.
<i>Clusters (Guard Ring)</i>	Create a Guard Ring for any cluster.
<i>Modgens (Guard Ring)</i>	Create a Guard Ring for any modgen.
<i>Same Device Type (Orientation)</i>	Group all primitive instances according to device type.
<i>Modgens (Orientation)</i>	Group all devices for the same modgen to set an orientation constraint.
<i>Inverters (Alignment)</i>	Group devices that form an inverter structure.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

<i>Inverters (Distance)</i>	Group devices that form an inverter structure.
<i>ALL</i>	Finds all placement constraints sequentially in the order of their listing in the <i>Search for</i> list.

Routing Constraints

All Finders related to Routing Constraints are detailed below:

Routing Finders	Description
<i>Supply Nets (Net Priority)</i>	Creates a group for each supply net.
<i>Bus and Bundle (Bus)</i>	Creates a bus for all signals of a bus net.
<i>Bus Bits</i>	Creates a bus for each collection of bus bits.
<i>ALL</i>	Finds all routing constraints sequentially in the order of their listing in the <i>Search for</i> list.

Clusters

All Finders related to Clusters are detailed below:

Clusters Finders	Description
<i>Common Bulk (Cluster)</i>	Group devices with same cell name and a common bulk.
<i>Same Cell and Size (Cluster)</i>	Group primitive devices with same cell name and size.
<i>Same Cell, Finger Width MOS (Cluster)</i>	Group active devices with same cell name, bulk connection and finger width.
<i>Digital Cells (Cluster)</i>	Group all standard cells together. To be captured, all standard cells must be registered with the function: <pre>ciRegisterDevice("standardCell" list(list("libName" "cellName" "viewName") list(LCV2) ...)</pre>
<i>ALL</i>	Finds all clusters sequentially in the order of their listing in the <i>Search for</i> list.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

Matched Parameters

All [Finders](#) related to Matched Parameters are detailed below:

Matched Parameters Finders	Description
<i>Symmetric Pairs (Matched Parameters)</i>	Group instances by symmetrical pairs according to connectivity to match their parameters.
<i>Same Cell, Size, Bulk Devices (Matched Parameters)</i>	Group devices with same cell name, bulk connections and size (w, l, r, c)
<i>Same Cell, Finger Width MOS (Matched Parameters)</i>	Group devices with same cell name and finger width (fw)
<i>Current Mirror (Matched Parameters)</i>	Group devices of a Current Mirror to set a Matched Parameters constraint.
<i>ALL</i>	Finds all matched parameters sequentially in the order of their listing in the <i>Search for</i> list.

Modgens

All [Finders](#) related to Modgens are detailed below:

Modgens Finders	Description
<i>Same Cell, Size, Bulk Devices (Modgen)</i>	Group devices with same cell name, bulk, connection and size (w, l, r, c).
<i>Digital Cells (Modgen)</i>	Group all standard cells together. To be captured, all standard cells must be registered with the function: <pre>ciRegisterDevice("standardCell" list(list("libName" "cellName" "viewName") list(LCV2) ...)</pre>
<i>ALL</i>	Finds all modgens sequentially in the order of their listing in the <i>Search for</i> list.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

Orientations

The *Orientations* category allows for setting orientations on instances either per device or as in schematic.

All finders in the *Orientations* category are detailed below:

Orientations Finders	Description
<i>Orientation as in Schematic</i>	Group all primitive instances according to schematic orientation.
<i>Same Device Type (Orientation)</i>	Group all primitive instances according to device type.
<i>Modgens (Orientations)</i>	Group all devices for the same modgen to set an orientation constraint.
<i>ALL</i>	Finds all orientations sequentially in the order of their listing in the <i>Search for</i> list.

Important

The `maxMatchedPerDevice` environment variable is added to control the maximum number of matches that a device may appear in when the Circuit Prospector structure finders are run. For example, Diff Pair and Current Mirror. This environment variable takes the integer value that defaults to 1.

Symmetries

All Finders related to Symmetries are detailed below:

Symmetries Finders	Description
<i>Instance (Symmetry By Pre-Selection)</i>	Group pairs of instances that are geometrically symmetric about the centre of the current selected set. Note: Instances in geometrically symmetric regions of the schematic need to be selected prior to running this finder.
<i>Instance (Symmetry By Connectivity)</i>	Group pairs of instances that are determined to be symmetric based on their connectivity.
<i>Nets (Symmetry By Connectivity)</i>	Group pairs of nets that are determined to be symmetric based on their connectivity.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

<i>Pins (Symmetry By Connectivity)</i>	Group pairs of pins that are determined to be symmetric based on their connectivity.
<i>ALL</i>	Finds all symmetries sequentially in the order of their listing in the <i>Search for</i> list.

Analog Device Structures

All [Finders](#) related to Analog Device Structures are detailed in the following table.

These finders are used to create either device groups or Modgens in the Virtuoso automated device placement and routing flow. For more information about this flow, see [Virtuoso Automated Device Placement and Routing Flow Guide](#).

Analog Device Structures Finders	Description
<i>APR Device Group from DI</i>	Groups APR Group Design Intent structures that are set in schematic.
<i>APR Differential Pair</i>	Groups devices that form a differential pair structure.
<i>APR Differential Pair - Cross Coupled</i>	Groups devices that form a cross-coupled differential pair structure.
<i>APR Current Mirror</i>	Groups devices that form a current mirror structure.
<i>APR Cascode Current Mirror</i>	Groups devices that form a cascoded current mirror structure.
<i>APR Cascode Series Current Mirror</i>	Groups devices that form a cascoded series current mirror structure.
<i>APR Cascode</i>	Groups cascoded MOS transistor structures.
<i>APR Active Same Size Common Gate</i>	Groups same-sized devices that share the same gate connection.
<i>APR Active Same Size Common Gate and Source</i>	Groups same-sized devices with the same gate and source connections.
<i>APR Active Same Size Common Source</i>	Groups same-sized devices with the same source connection.
<i>APR Active Same Cell M-Factored Device</i>	Groups active devices with mfactor.
<i>APR Active Same Cell Stacked Device</i>	Groups transistors in series stacks.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

<i>APR Active Same Cell Iterated Device</i>	Groups iterated devices.
<i>APR Active Same Cell and Size</i>	Groups active devices with same cell name and size.
<i>APR Passive Same Cell and Size</i>	Groups passive devices with same cell names or the same cell sizes and values.
<i>APR User Defined Group</i>	Groups device groups inside text boxes.
<i>APR Instances (Symmetry By Connectivity)</i>	Groups pairs of instances that are determined to be symmetric based on their connectivity.
<i>APR Instances (Symmetry By Connectivity with common source or drain)</i>	Groups pairs of instances that are determined to be symmetric based on their connectivity with a common source or drain.
<i>APR Nets (Symmetry By Connectivity)</i>	Groups nets that are determined to be symmetric based on their connectivity.
<i>APR Pins (Symmetry By Connectivity)</i>	Groups pins that are determined to be symmetric based on their connectivity.

Creating and Editing Categories

You can create a new category of finders, or edit existing categories, using the Edit Categories form accessible from the “...” option adjacent to the *Category* drop-down in the *Circuit Prospector* assistant.

Additionally, from here, any finder can be added or removed to/from any category, and the order of the finders can be changed for any category.

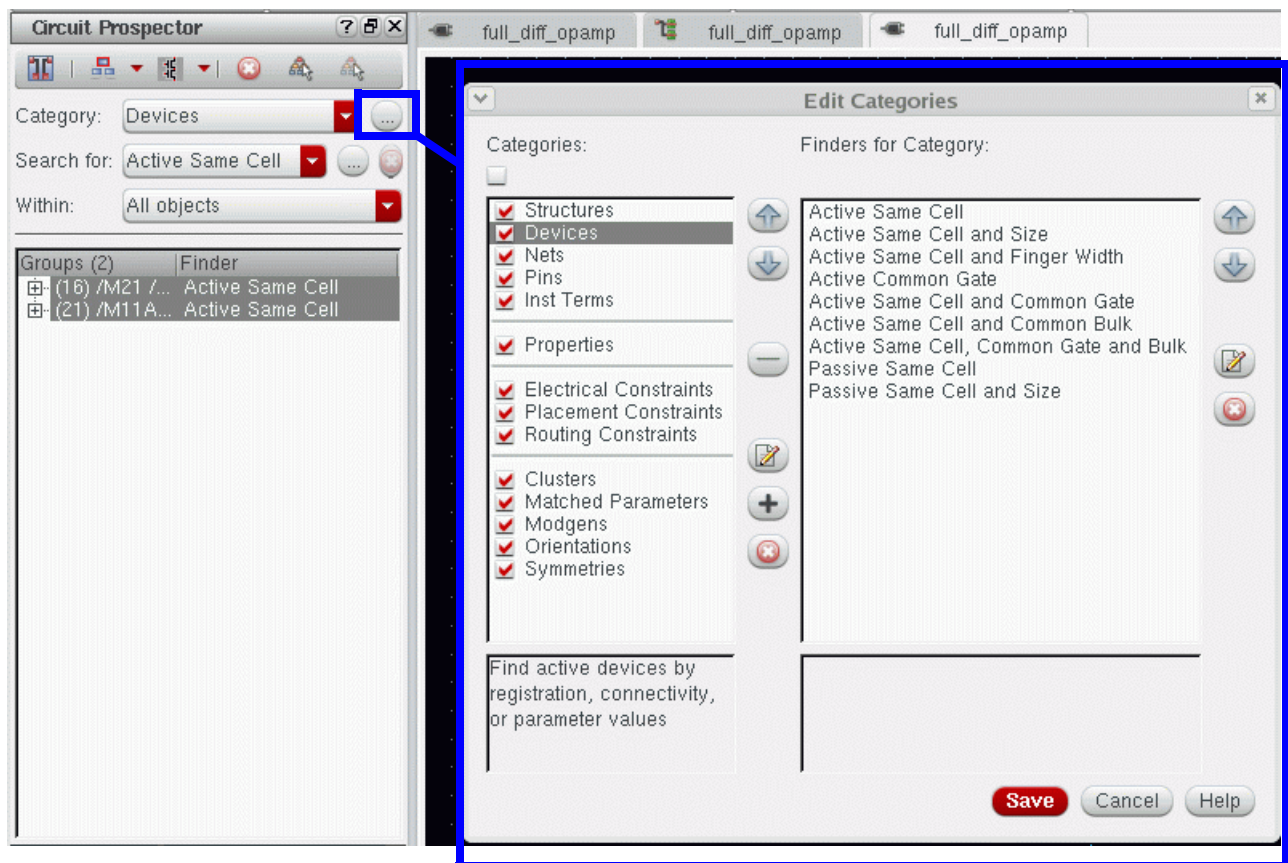


Figure 2-12 The Edit Categories Form

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

Figure 2-13

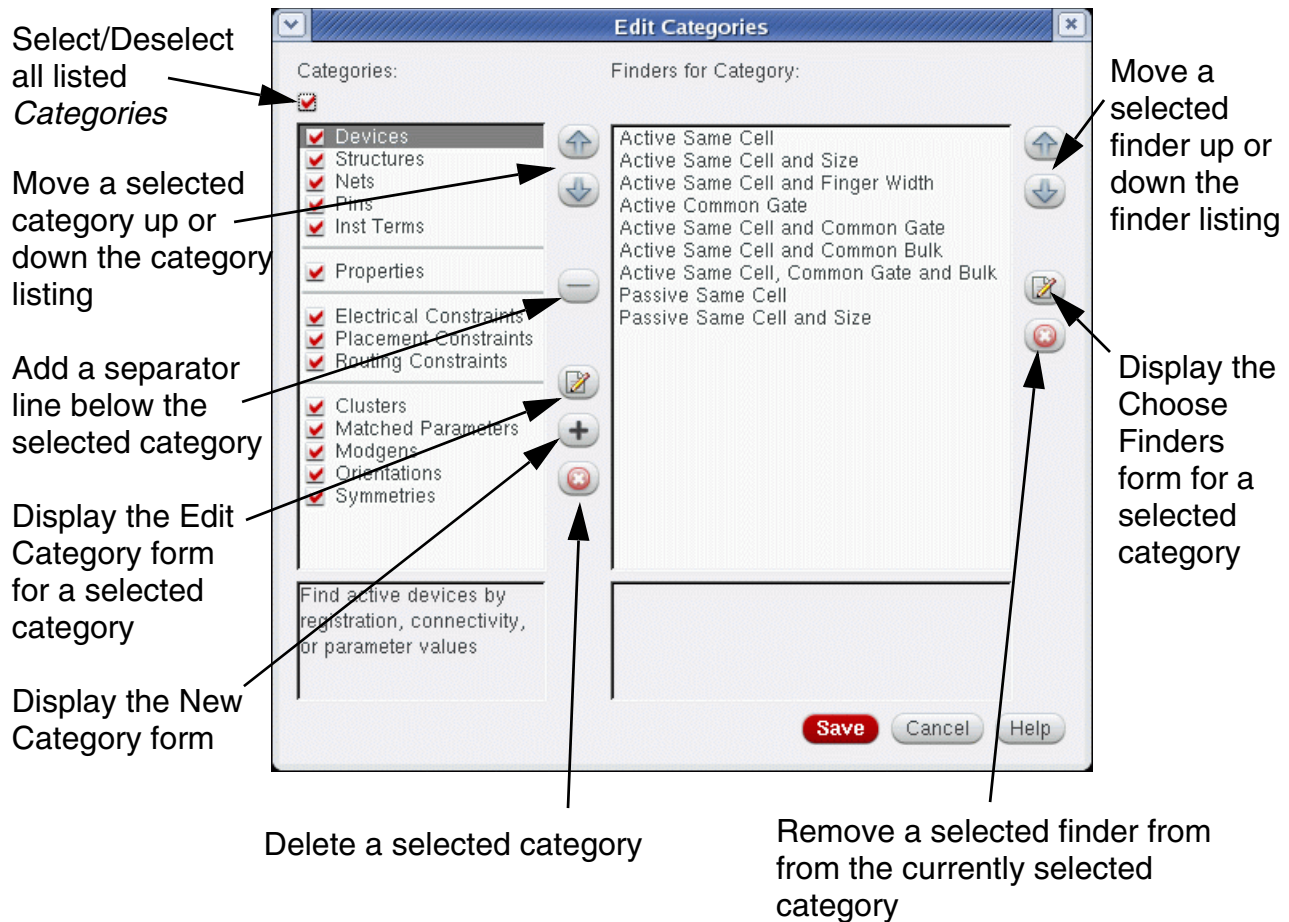
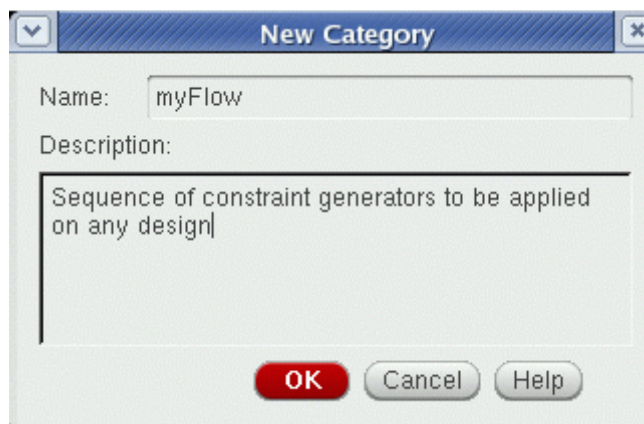


Figure 2-14 The Edit Categories Form Use Model

Adding a New Category

To add a new category to the *Circuit Prospector* assistant:

1. Select the “...” option adjacent to the *Category* drop-down list box.
This will display the Edit Categories form.
2. Choose the *Add New Category* (“+” icon) option in the Edit Categories form.
This will display the New Category form.



3. Enter a new category *Name*.
4. Optionally, add a *Description* for the new category.
5. Click the *OK* button. The new category will be added to the *Categories* list in the Edit Categories form.
6. Click the *Save* button, in the Edit Categories form, to update the category and finder information to allow access to them from the *Circuit Prospector* assistant.

Editing an Existing Category

To edit the name and/or description of a category:

1. Select the “...” option adjacent to the *Category* drop-down list box.
This will display the Edit Categories form.
2. Select the category to be edited from the *Categories* listing.
3. Choose the *Edit Category* (icon) option in the Edit Categories form.
This will display the Edit Category form.
4. Edit the *Name* and/or *Description* of the currently selected category.
5. Click the *OK* button. The category name and/or description will be updated and displayed in the Edit Categories form.
6. Click the *Save* button, in the Edit Categories form, to update the category and finder information to allow access from the *Circuit Prospector* assistant.

Adding and Removing Finders from a Category

To add or remove finders to/from a new or existing category:

1. Select the “...” option adjacent to the *Category* drop-down list box for you category that you want to edit finder content for.

This will display the Edit Categories form.

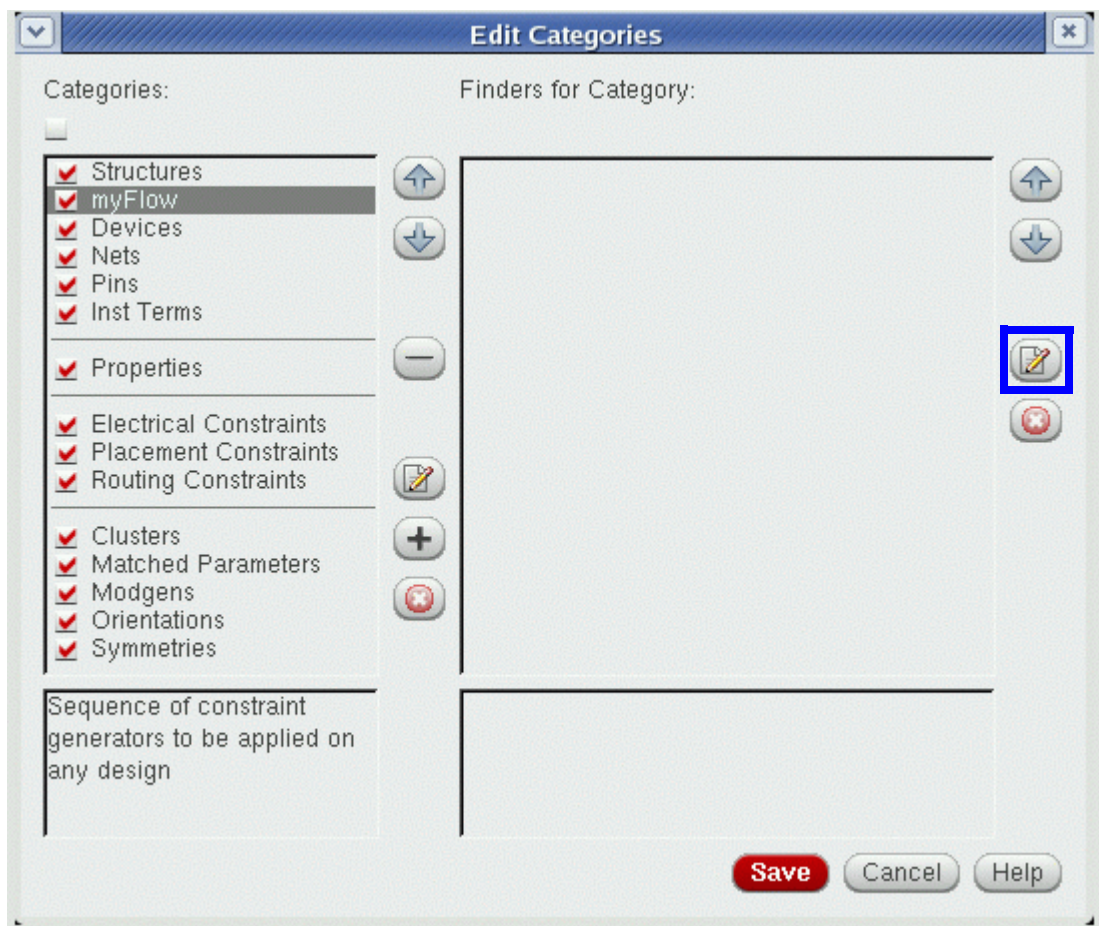


Figure 2-15 myFlow Category - Currently Contains No Finders

2. Select the category whose finders are to be edited from the *Categories* listing.
3. Choose the *Choose Finders for Category* (icon) option in the Edit Categories form.

This will display the Choose Finders form.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

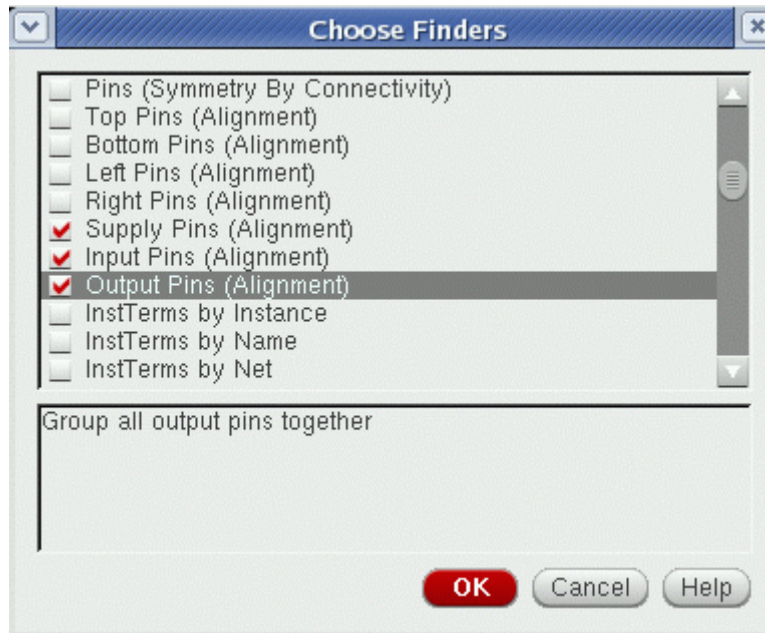


Figure 2-16 The Choose Finders Form

4. Select and/or deselect the finders that you want to include in the selected category.
5. Click the *OK* button.

The *Finders for Category* listing will be updated to reflect the changes made.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

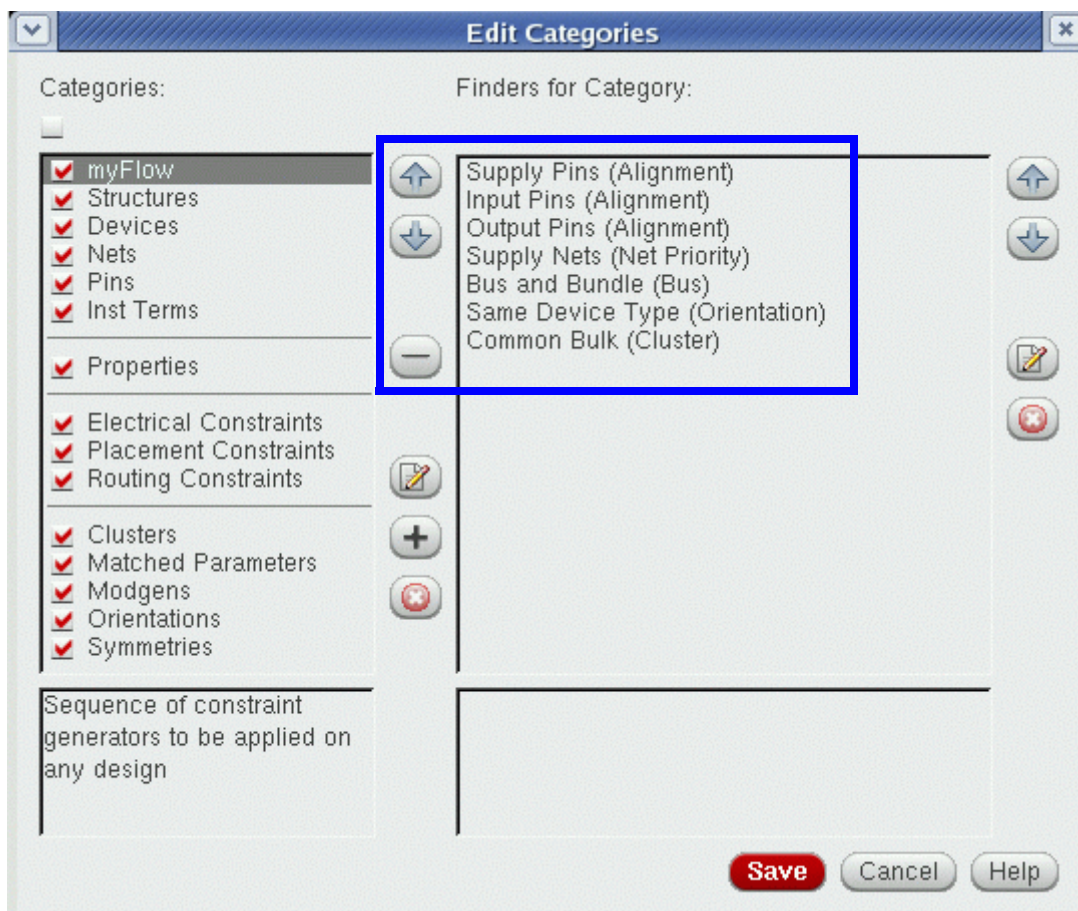


Figure 2-17 New Finders Added for myFlow Category

6. Click the *Save* button, in the *Edit Categories* form, to update the category and finder information to allow access from the *Circuit Prospector* assistant.

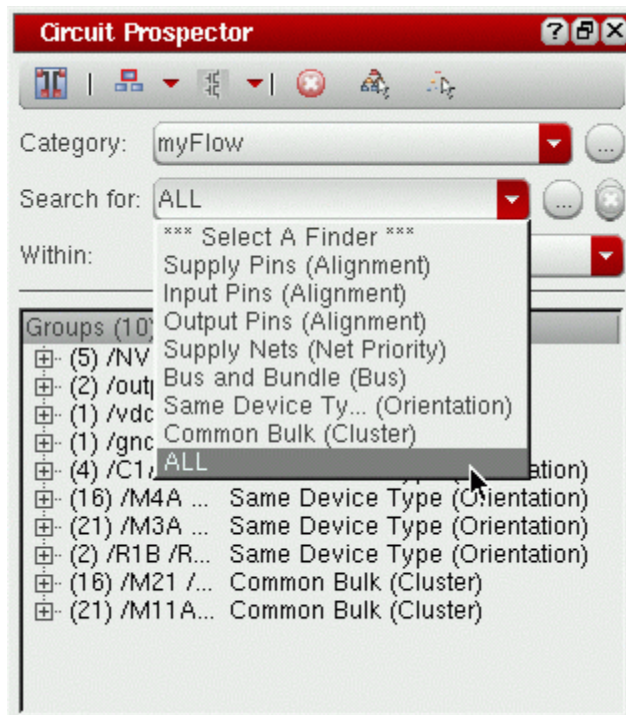


Figure 2-18 myFlow Category with Finders

Detecting Current Paths and Creating Constraints

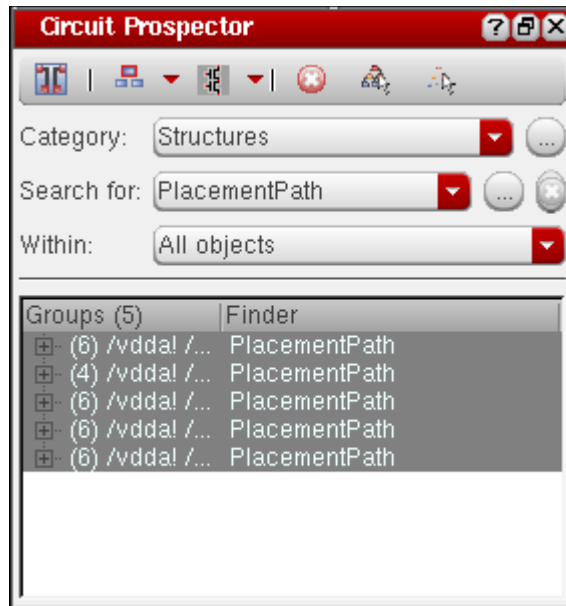
To use the *Circuit Prospector* assistant to detect current paths and create constraints in a schematic cellview:

1. Display the *Circuit Prospector* assistant by selecting *Window – Assistants – Circuit Prospector*.
2. Select the *Structures* circuit prospector category.

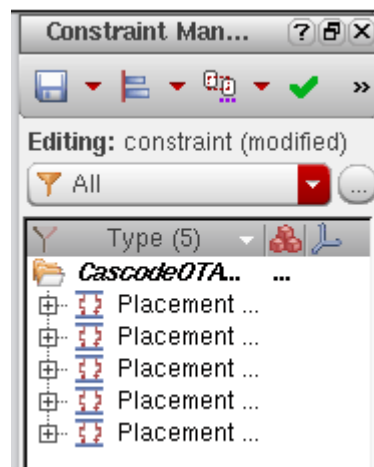
Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

3. Select the *Placement Path* finder to search for relevant design objects. The results (objects or groups of objects that meet the search criteria) are retrieved automatically and displayed in the results table.



4. Select the objects to which the constraint needs to be applied. This is an optional step.
5. Select the  button to add the Placement Path constraint to the selected objects. The new constraints are added to the *Constraint Manager* table.



For more information on how to use the *Circuit Prospector* assistant, see [The Circuit Prospector Constraint Creation Flow](#).

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

For more information on how to use the Constraint Manager, see [Chapter 1, “The Constraint Manager Assistant.”](#)

Finders

Finders are search algorithms that search for devices, pins, and nets that meet specific, customizable, search criteria. For example, common circuit structures, related devices (of the same size, area, or type), devices with certain characteristics or shared connectivity, or perhaps symmetrical devices with particular properties.

The *Circuit Prospector* provides a set of pre-defined finders that can be listed by clicking the *Search for* drop-down list box. New finders can be created, current finders edited, and no longer required finders deleted.

See also:

- [Creating Finders](#)
- [Editing Finders](#)
- [Deleting Finders](#)

For information on pre-defined finders, the following sections on pre-defined categories each provide descriptions of the finders they contain.

- [Rapid Analog Prototype](#) on page 302
- [Devices](#) on page 350
- [Structures](#) on page 338
- [Nets](#) on page 354
- [Pins](#) on page 354
- [Inst Terms](#) on page 355
- [Properties](#) on page 356
- [Electrical Constraints](#) on page 357
- [Placement Constraints](#) on page 357
- [Routing Constraints](#) on page 358
- [Symmetries](#) on page 360
- [Matched Parameters](#) on page 359
- [Clusters](#) on page 358
- [Modgens](#) on page 359

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

- [Orientations](#) on page 360

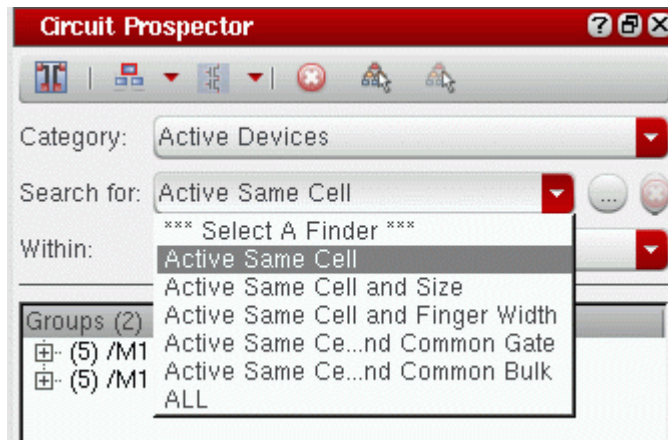


Figure 2-19 Pre-defined finders

Creating Finders

Note: See also [Creating New Finders](#) in the *Virtuoso Unified Custom Constraints Configuration Guide*.

You can edit the settings of a current finder to create a new finder, based on existing and/or new settings.

1. Click the ... (ellipsis) *Edit Finder* button to the right of the *Search for* drop-down list box in the *Circuit Prospector* assistant.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

This will display Edit Finder Form.

The screenshot shows the 'Edit Finder' dialog box. It has a title bar with the text 'Edit Finder'. Inside, there are several sections: 'Name:' with a text field containing 'Active Same Cell'; 'Description:' with a large text area containing 'Group active devices with same cell name'; 'Search Using Iterator:' with a dropdown menu showing 'Same Cell Iterator'; 'Match Expression:' with a text area containing a SKILL expression: `cilsDevice(device "fet") || cilsDevice(device "bjt") || cilsDevice(device "diode")`; 'Properties:' with a list of properties: 'pin# (various)', 'instNamePr... M', 'Model Name (various)', 'Multiplier (various)', 'Length (various)', 'Total Width (various)', 'Finger Width (various)', and 'Fingers (various)'; and 'Default Generator or Action:' with a dropdown menu showing 'Same Cell Name - Group + Relative Orientation'. At the bottom right are 'Save', 'Cancel', and 'Help' buttons.

Figure 2-20 The Edit Finder form

2. Enter the name of the new finder in the *Name* field.

3. Enter a *Description* of the new finder.

This text entered here will be displayed as a tooltip when you float your cursor over the finder when it is listed in the *Search for* drop-down list box.

4. In the *Search Using Iterator* drop-down list box, choose the iterator from the drop-down list that you want the new finder to use when performing a search.

Note: Clicking the ... (ellipsis) button to the right of the *Search Using Iterator Generator* will invoke Edit Iterator Form where you can add, edit, or delete iterators. For more information see Iterators.

5. In the *Matching Expression* section specify a SKILL expression for the chosen iterator to apply when it iterates over cellview objects.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

The *Matching Expression* field can call a number of pre-defined functions, for example `ciIsDevice(device "fet")`, to filter the type of objects found.

If an expression returns a positive value the current set of iterated objects will be collected and returned once the iteration is complete. The current set of iterated objects may be a single object or a set of objects, dependent upon the iterator selected.

To the right of the expression entry area is a property display area which lists all of the properties of the currently selected objects in the design canvas.

Note: Right-clicking over this table will display a context-menu which enables you to rearrange the table display. For example, you can choose to *View by Property* or *View by Object*, *Expand All* or *Collapse All* of the properties displayed, or choose to move to the *Next* or *Previous* selected property.

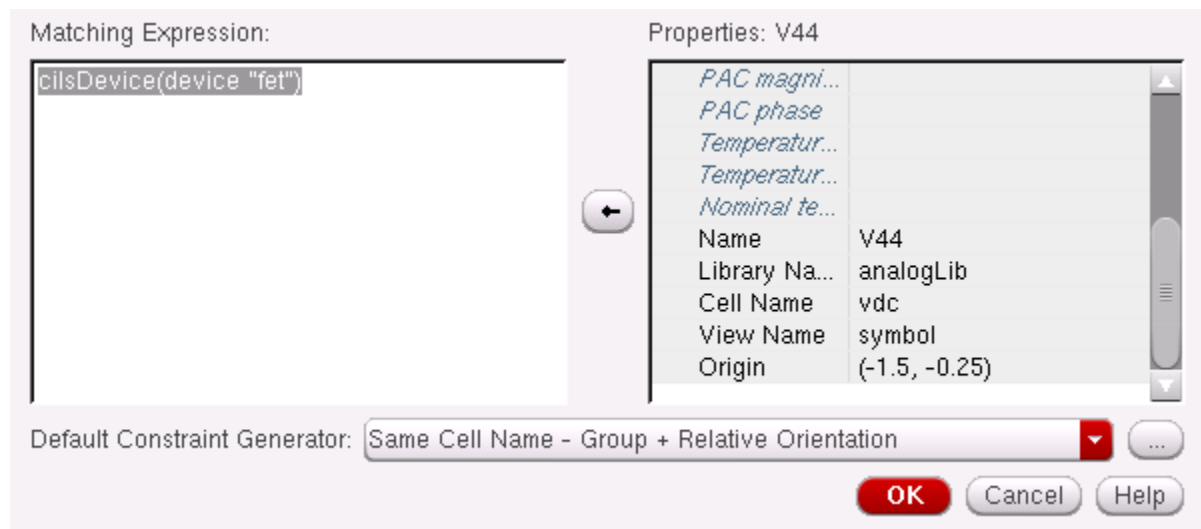


Figure 2-21 Matching Expression section of the Edit Finder form

6. Select a *Default Constraints Generator* for the new finder to use.

Once the finder has been run the selected constraint generator will be used when *Generate Constraints* on the *Circuit Prospector* toolbar is selected.

Note: Clicking the ... (ellipsis) button to the right of the *Default Constraints Generator* section will invoke *Edit Generator or Action Form* where you can add, edit, or delete constraint generators. For more information see *Constraint Generators*.

7. Click the *OK* button to create the new finder. The new finder will be added to the list of finders available under the *Search for* drop-down list box.

Editing Finders

Note: If you edit an existing pre-defined finder, then the modified finder will be stored in the local `.cadence` directory (`.cadence/dfII/ci/finders/<finderName>.il`) and will override the original pre-defined finder.

1. Click the ... (ellipsis) *Edit Finder* button to the right of the *Search for* drop-down list box in the *Circuit Prospector* assistant.

This will display Edit Finder Form.

2. Optionally, edit the *Description* of the finder.
3. Optionally, click the ... (ellipsis) button to the right of the *Search Using Iterator* drop-down list box to invoke Edit Iterator Form.

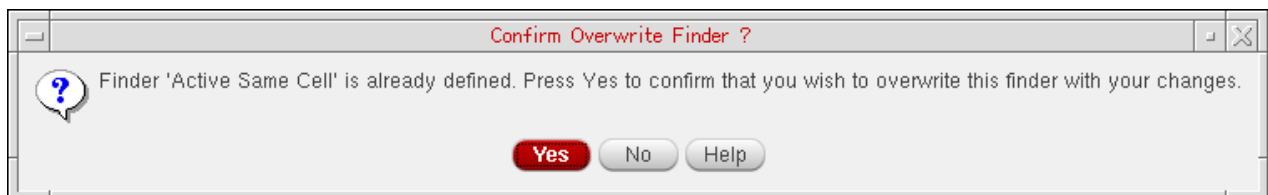
Here, you can add, edit, or delete the iterator associated with the finder. For more information see Iterators.

4. Optionally, in the *Matching Expression* section, edit the SKILL expression which the iterator applies as it iterates over cellview objects.

Note: If an expression returns a positive value the current set of iterated objects will be collected and returned once the iteration is complete. The current set of iterated objects may be a single object or a set of objects, dependent upon the iterator selected.

Important

If you make any changes in the SKILL expression and click the *Save* button, the *Confirm Overwrite Finder?* dialog box is displayed asking for the confirmation to override the SKILL expression, as shown below.



5. Optionally, change the *Default Constraints Generator* to be used.

Note: Clicking the ... (ellipsis) button to the right of the *Default Constraints Generator* section will invoke Edit Generator or Action Form where you can add, edit, or delete constraint generators. For more information see Constraint Generators.

6. Click the *OK* button to apply the finder edits.

Deleting Finders

To delete a finder:

1. Select the finder you want to delete from the *Search for* drop-down list box in the *Circuit Prospector* assistant.
2. Click the adjacent *Delete Finder* (cross) button to delete the selected finder.

A window message asking you to confirm deletion is displayed.

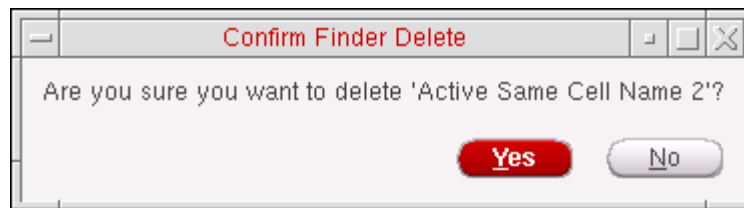


Figure 2-22 Confirming the Deletion of a Finder

Note: You cannot delete a pre-defined finder.

3. Select *Yes* to confirm finder deletion, or select *No* to cancel finder deletion.

The finder will be removed from the *Search for* drop-down list box.

Iterators

Iterators are SKILL functions that analyze cellview devices, nets, and pins in a variety of ways, for example iterate all selected objects, all instances, or all symmetrical devices.

As already described, Finders are wrappers around iterators that provide a SKILL expression for an iterator to apply so that it can return a list of objects that will satisfy that expression. In this way, iterators can be used by a multitude of finders.

Note: Iterators can also have their own internal filtering settings.

This section contains information on:

- Pre-Defined Iterators on page 380
- Creating Iterators on page 383
- Editing Iterators on page 385
- Deleting Iterators on page 386

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

Pre-Defined Iterators

You can view a list of the pre-defined iterators by clicking the *Search Using Iterators* drop-down option in [Edit Finder Form](#).

Note: To view, add, or edit iterators click the ... (ellipsis) button to the right of the *Search Using Iterators* option. This will displays [Edit Iterator Form](#).

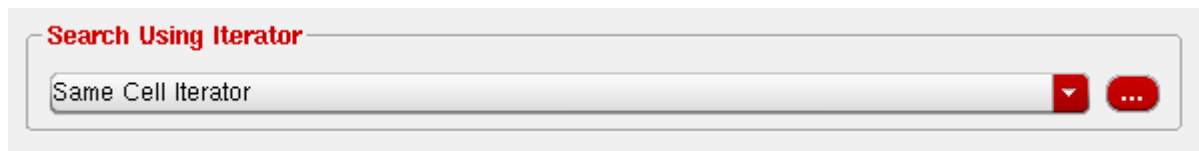


Figure 2-23 Search Using Iterator drop-down list box of Edit Finder form

Pre-Defined Iterator	Description	Iterator SKILL Function Name
<i>Same Cell Iterator</i>	Iterate over all devices with the same cell name.	ciSameCellIterator
<i>Pin Iterator</i>	Iterate over all pins.	ciPinIterator
<i>Net Iterator</i>	Iterate over all nets.	ciNetIterator
<i>Instance Iterator</i>	Iterate over all instances. Variable “device” can be used in matched expression.	ciInstIterator
<i>InstTerm Iterator</i>	Iterate over all instance terminals. Variables “instTerm” and “inst” can be used in matched expression.	ciInstTermIterator
<i>XY Symmetric Iterator</i>	Iterate over all X/Y symmetric active devices with the same cell name.	ciXYSymmetricIterator
<i>XY Net Symmetric Iterator</i>	Iterate over all X/Y symmetric nets.	ciXYNetSymmetricIterator
<i>MOS Cascode</i>	Iterate over all cascoded MOS transistor structures	ciMOSCascodedIterator
<i>MOS Current Mirror Iterator</i>	Iterate over all MOS Current Mirrors.	ciMOSCurrentMirrorStructIterator
<i>MOS Cascoded Current Mirror Iterator</i>	Iterate over all MOS Cascoded Current Mirrors.	ciMOSCascodedCurrentMirrorStructIterator

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

<i>MOS Cascoded Current Mirror Iterator2</i>	Iterate over all MOS Cascoded Current Mirrors - with at least one of the leading pairs being diode connected.	<code>ciMOSCascodedCurrentMirrorStructIterator2</code>
<i>MOS Transmission Gate Iterator</i>	Iterate over all MOS Transition Gates.	<code>ciMOSTransmissionGateStructIterator</code>
<i>MOS Differential Pair Iterator</i>	Iterate over all MOS Differential Pairs.	<code>ciMOSDifferentialPairStructIterator</code>
<i>MOS Common Gate Iterator</i>	Iterate over all MOS Common Gates.	<code>ciMOSCommonGateStructIterator</code>
<i>MOS Parallel Iterator</i>	Iterate over MOS Parallels.	<code>ciMOSParallelStructIterator</code>
<i>MOS Active Load Iterator</i>	Iterate over all MOS Active Loads.	<code>ciMOSActiveLoadStructIterator</code>
<i>MOS Inverter Iterator</i>	Iterate over all MOS Inverters.	<code>ciMOSInverterStructIterator</code>
<i>MOS Cross Coupled Quad Iterator</i>	Iterate over all MOS Cross Coupled Quads.	<code>ciMOSCrossCoupledQuadStructIterator</code>
<i>XY Instance Symmetric Iterator</i>	Iterate over all X/Y symmetric devices with the same cell name.	<code>ciXYInstSymmetricIterator</code>
<i>Signal Iterator</i>	Iterates over all signals. Variable “signals” can be used in matchExpr.	<code>ciSignalIterator</code>
<i>Bundle Signal Iterator</i>	Iterate over all bundle nets and return the list of corresponding signals. Variable “bundleSignals” can be used in matchExpr.	<code>ciBundleSignalIterator</code>
<i>XY Pin Symmetric Iterator</i>	Iterate over all X/Y symmetric pins.	<code>ciXYPinSymmetricIterator</code>

The above information is also displayed in the [Edit Iterator Form](#).

Iterator SKILL Functions

The iterator SKILL function is called to perform an iteration. The function takes `CellView` (object ID) and `matchExpressions` (string of characters) as arguments, where `CellView` is the current cellview and `matchExpression` is the *Matching Expression* (see [Edit Finder Form](#)) that was defined for the finder that will call the iterator.

The function returns a list of object lists that satisfy `matchExpression`, while iterating through the passed `CellView`.

Creating Iterators

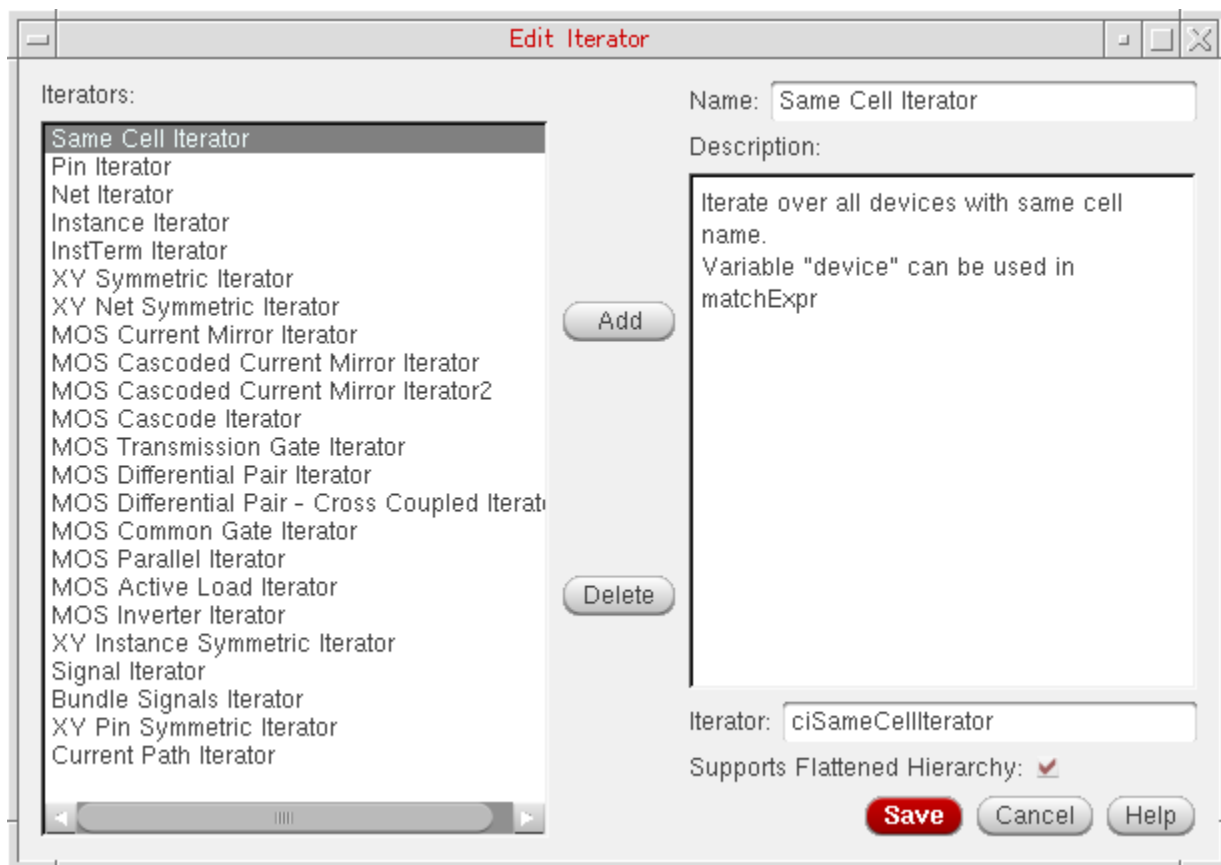
To create a new iterator:

1. Click the ... (ellipsis) button to the right of the *Search for* drop-down list box in the *Circuit Prospector* assistant.

This displays Edit Finder Form.

2. Click the ... (ellipsis) button to the right of the *Search Using Iterator* drop-down list box in the Edit Finder form.

This displays Edit Iterator Form.



3. Replace the name of the iterator in the *Name* field to create a new iterator name.

Note: Editing the name of the current iterator will not remove it from the *Iterators* list.

4. Edit the *Description* of the new iterator.
5. Edit the *Iterator* SKILL function name for the new iterator.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

The *Iterator* SKILL function name is the name of the SKILL function that is to be called to perform the iteration. The function takes a cellview (as an object ID) and a matching expression (as a string of characters) as arguments, where the cellview is the current cellview and the matching expression is the matching expression that has been defined for the finder that will call this iterator (see [Creating Finders](#)). The function returns a list of object lists that satisfy the matching expression while iterating through the passed cellview.

6. Specify whether or not the iterator *Supports Flattened Hierarchy*.
7. Select the *Add* button to add the new iterator to the *Iterators* list.
8. Repeat steps 3-7 as necessary to add further iterators.
9. Click the *OK* button to accept the iterator additions you have made.

The Edit Finder form will again be displayed and any new iterators will be included in the *Search Using Iterator* drop-down list box.

Note: The iterator calls a number of pre-defined helper functions, for example `ciIsDevice(device "fet")`, to filter the type of objects found.

Editing Iterators

To edit a current iterator:

1. Click the ... (ellipsis) button to the right of the *Search for* drop-down list box in the *Circuit Prospector* assistant.

This displays Edit Finder Form.

2. Click the ... (ellipsis) button to the right of the *Search Using Iterator* drop-down list box in the Edit Finder form.

This displays Edit Iterator Form.

3. Select the iterator you want to edit from the *Iterators* list.
4. Optionally, edit the *Description* of the selected iterator.
5. Optionally, edit the *Iterator SKILL* function name of the selected iterator.
6. Optionally, edit the *Supports Flattened Hierarchy* of the selected iterator.
7. Repeat steps 3-6 as necessary to edit further iterators.
8. Click the *OK* button to accept the edits you have made.

The Edit Finder form will again be displayed with iterator edits applied.

Deleting Iterators

Note: Care should be taken as you can delete pre-defined iterators.

To delete an iterator:

1. Click the ... (ellipsis) button to the right of the *Search for* drop-down list box in the *Circuit Prospector* assistant.

This displays Edit Finder Form.

2. Click the ... (ellipsis) button to the right of the *Search Using Iterator* drop-down list box in the Edit Finder form.

This displays Edit Iterator Form.

3. Select the iterator you want to delete from the *Iterators* list.
4. Click the *Delete* button.
5. Repeat steps 3-4 as necessary to delete further iterators.
6. Click the *Cancel* button to apply the iterator deletions and return to the Edit Finder form.

The iterators that you have deleted will no longer be listed in the *Search Using Iterator* drop-down list box.

Constraint Generators

Constraint generators are used to generate constraints for those groups of nets, instances, or pins that are selected in the results section of the *Circuit Prospector* assistant.

Note: Nets are listed in the *Circuit Prospector* results table for all default constraint generators, as there may be default constraints associated with these nets that need to be considered.

A constraint generator will generate one or more constraints with either default or specific parameter values.

Constraint generators might need to access those object properties that are to be constrained (for example, the object's orientation). As the associated finder may have been run hierarchically, the object may then be located in either the current cellview or a lower level cellview. Therefore, to get the database ID (dbID) of the object use:

```
ciFindObjectInHier(fullObjectName cache 'objType)
```

To ensure that the object is searched for at the correct hierarchical level use, for example:

```
ciFindObjectInHier("/I1/I2/MN1" cache 'inst)
```

A set of pre-defined constraint generators are provided along with the capability of creating new generators. For more information, see:

- [Pre-Defined Constraint Generators](#) on page 388
- [Creating Constraint Generators](#) on page 396
- [Editing Constraint Generators](#) on page 398
- [Deleting Constraint Generators](#) on page 399

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

Pre-Defined Constraint Generators

The *Circuit Prospector* assistant provides pre-defined constraint generators for each base constraint type, plus a selection of variations with alternative parameter settings.

Pre-defined Constraint Generator	Description	Expression
<i>Matched Parameters</i>	Generate Matched Parameters	<code>ciConCreateExpanded(cache 'matchedParameters ?members append(inst pins) ?verbose nil)</code>
<i>Matched Size</i>	Generate Matched Parameters for l and w	<code>ciConCreateExpanded(cache 'matchedParameters ?members append(insts pins ?params list(list("matchSubset" "l w"))) ?verbose nil)</code>
<i>Symmetry</i>	Generate Symmetry with vSymm axis.	<code>ciConCreateExpanded(cache 'symmetry ?members append(insts append(pins instTerms)) ?axis "vSymm" ?verbose nil)</code>
<i>vSymmetry</i>	Generate Symmetry with vSymm axis.	<code>ciConCreateExpanded(cache 'symmetry ?members append(insts append(pins instTerms)) ?axis "vSymm" ?verbose nil)</code>
<i>hSymmetry</i>	Generate Symmetry with hSymm axis.	<code>unless(ciAxisExists(cache "hSymm") ciAxisCreate(cache "hSymm" list(list("direction" "horizontal")))) ciConCreateExpanded(cache 'symmetry ?members append(insts append(pins instTerms)) ?axis "hSymm" ?verbose nil)</code>
<i>Alignment</i>	Generate an alignment constraint	<code>ciConCreateExpanded(cache 'alignment ?members append(insts append(pins instTerms)) ?verbose nil)</code>
<i>Orientation</i>	Generate an orientation constraint	<code>ciConCreateExpanded(cache 'orientation ?members append(insts pins ?verbose nil)</code>
<i>Module</i>	Generate a modgen constraint	<code>ciConCreateExpanded(cache 'modgen ?members insts ?verbose nil)</code>
<i>Same Cell Name - Group + Relative Orientation</i>	Generate Cluster and Same Orientation Constraints	<code>when(cadr(instsNetsPins) ciConCreateExpanded(cache 'cluster ?members insts ?note strcat("CP: Group together devices with same cell name") ?verbose nil) && ciConCreateExpanded(cache 'relativeOrientation ?members insts ?note strcat("CP: Matching orientation for devices with same cell name") ?verbose nil)</code>

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

<i>Common Gate - Group + Relative Orientation</i>	Generate Cluster and Same Orientation Constraints	<pre> when(cadr(instsNetsPins) ciConCreateExpanded(cache 'cluster ?members insts ?note strcat("CP: Group together devices with common gate") ?verbose nil) && ciConCreateExpanded(cache 'relativeOrientation ?members insts ?note strcat("CP: Matching orientation for devices with common gate") ?verbose nil) </pre>
<i>Same Well - Group + Relative Orientation</i>	Generate Cluster and Same Orientation Constraints	<pre> when(cadr(instsNetsPins) ciConCreateExpanded(cache 'cluster ?members insts ?note strcat("CP: Group together devices with same well") ?verbose nil) && ciConCreateExpanded(cache 'relativeOrientation ?members insts ?note strcat("CP: Matching orientation for devices with common gate") ?verbose nil) </pre>
<i>Module + Matched Parameters</i>	Generate Modgen and Matched Parameters	<pre> ciConCreateExpanded(cache 'modgen ?members insts ?note strcat("CP: Module for devices of MOS cross coupled quad") ?verbose nil) && ciConCreateExpanded(cache 'matchedParameters ?members insts ?note strcat("CP: Matched parameters for devices of MOS cross coupled quad") ?verbose nil) </pre>
<i>Group + Relative Orientation + Vertical Alignment</i>	Generate Cluster, Same Orientation and Alignment with vAlign Axis	<pre> unless(ciAxisExists(cache "vAlign") ciAxisCreate(cache "vAlign" list(list("direction" "vertical"))))) ciConCreateExpanded(cache 'cluster ?members insts ?note strcat("CP: Group together NMOS and PMOS of transmission gate or inverter") ?verbose nil) && ciConCreateExpanded(cache 'relativeOrientation ?members insts ?note strcat("CP: Matched orientation of NMOS and PMOS of transmission gate or inverter") ?verbose nil) && ciConCreateExpanded(cache 'alignment ?members insts ?axis "vAlign" ?params list(list("side" "center")) ?note strcat("CP: Align vertically NMOS and PMOS of transmission gate or inverter") ?verbose nil) </pre>

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

<i>Group + Relative Orientation + Horizontal Alignment</i>	Generate Cluster, Same Orientation and Alignment Constraints	<pre> when(cadr(instsNetsPins) ciConCreateExpanded(cache 'cluster ?members insts ?note strcat("CP: Group together parallel devices or devices of MOS current mirror") ?verbose nil)) && ciConCreateExpanded(cache 'relativeOrientation ?members insts ?note strcat("CP: Matched orientation for parallel devices or devices of MOS current mirror") ?verbose nil) && ciConCreateExpanded(cache 'alignment ?members insts ?note strcat("CP: Horizontal alignment for parallel devices or devices of MOS current mirror") ?verbose nil) </pre>
<i>Group + Relative Orientation + Matched Parameters</i>	Generate Cluster, Same Orientation and Matched Parameters	<pre> when(cadr(instsNetsPins) ciConCreateExpanded(cache 'cluster ?members insts ?note strcat("CP: Group together devices with same cell name and size") ?verbose nil)) && ciConCreateExpanded(cache 'relativeOrientation ?members insts ?note strcat("CP: Matching orientation for devices with same cell name and size") ?verbose nil) && ciConCreateExpanded(cache 'matchedParameters ?members insts ?params list(list("matchSubset" nil)) ?note strcat("CP: Matched parameters for devices with same cell name and size") ?verbose nil) </pre>
<i>Symmetry + Matched</i>	Generate Symmetry with vSymm Axis and Matched Parameters	<pre> ciConCreateExpanded(cache 'symmetry ?members list(car(insts) cadr(insts)) ?axis "vSymm" ?note strcat("CP: Mirror symmetry for symmetrical devices") ?verbose nil) && ciConCreateExpanded(cache 'matchedParameters ?members append(insts pins) ?note strcat("CP: Matched parameters for symmetrical devices") ?verbose nil) </pre>
<i>Matched + Distance</i>	Generate Matched Parameters and Short Distance	<pre> ciConCreateExpanded(cache 'matchedParameters ?members append(insts pins) ?note strcat("CP: Matched parameters for devices of MOS differential pair") ?verbose nil) && ciConCreateExpanded(cache 'distance ?members append(insts pins) ?params list(list("max" 10.0)) ?note strcat("CP: Close proximity for devices of MOS differential pair") ?verbose nil) </pre>

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

<i>Matched + Symmetry + Distance</i>	Generate Matched Parameters, Symmetry with vSymm Axis and Distance	<pre>ciConCreateExpanded(cache 'matchedParameters ?members append(insts pins) ?verbose nil) && ciConCreateExpanded(cache 'symmetry ?members append(insts pins) ?axis "vSymm" ?verbose nil) && ciConCreateExpanded(cache 'distance ?members append(insts pins) ?verbose nil)</pre>
<i>Group</i>	Generate Cluster Constraint	<pre>ciConCreateExpanded(cache 'cluster ?members insts ?verbose nil)</pre>
<i>Supply Route Priority</i>	Generate High Priority Constraint	<pre>ciCreateRoutePriorityCon(cache nets 128)</pre>
<i>Non Supply Route Priority</i>	Generate Average Priority Constraint	<pre>ciCreateRoutePriorityCon(cache nets 64)</pre>
<i>Matching (strength)</i>	Ask for Matching Variants	<pre>ciRunMatchingConstraintsGenerator(args insts cache)</pre>
<i>Common Centroid</i>	Generate Common Centroid Constraints	<pre>(_ciCommonCentroidGen args insts cache)</pre>
<i>Inst Symmetry</i>	Generate Symmetry, Matched Parameters and Correlation Constraints	<pre>ciGeneratorForInstSymmetry(cache insts)</pre>
<i>Net Symmetry</i>	Generate Symmetry with vertSymm Axis and High Priority	<pre>ciGeneratorForNetSymmetry(cache nets)</pre>
<i>Same Orientation</i>	Generate Orientation Constraint per Schematic Orientation	<pre>let((inst) inst = substring(caar(insts) 2) while(nindex(inst "/") inst = substring(index(inst "/") 2)) ciConCreateExpanded(cache 'orientation ?params list(list("restrictTo" list(dbFindAnyInstByName(getEditCellView() inst)->orient))) ?members insts ?note strcat("CP: Same orientation as in schematic") ?verbose nil))</pre>
<i>Orientation per Device Type</i>	Generate Orientation Constraint per Device Type	<pre>let((inst orientL) inst = substring(caar(insts) 2) while(nindex(inst "/") inst = substring(index(inst "/") 2)) inst=dbFindAnyInstByName(getEditCellView() inst) orientL=list("R0" "MY") when(ciIsDevice(inst "standardCell") orientL=list("R0")) when(ciIsDevice(inst "nfet") orientL=list("R0" "MY")) when(ciIsDevice(inst "passive") orientL=nil) ciConCreateExpanded(cache 'orientation ?params list(list("restrictTo" orientL)) ?members insts ?note strcat("CP: Orientation per Device Type") ?verbose nil))</pre>

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

<i>Bus for Nets</i>	Generate Bus Constraint	<code>ciConCreateExpanded(cache 'bus ?members list(list(substring(caar(nets) 2) 'net)) ?verbose nil)</code>
<i>Bus for Signal Bits</i>	Generate Bus Constraint	<code>ciConCreateExpanded(cache 'bus ?members nets ?verbose nil)</code>
<i>sigType for registered nets</i>	Set sigType Property per Net Registration	<code>let ((net) net=dbFindNetByName(geGetEditCellView() substring(caar(nets) 2)) when(ciIsNet(net "power") net->sigType="supply") when(ciIsNet(net "ground") net->sigType="ground") when(ciIsNet(net "clock") net->sigType="clock") when(ciIsNet(net "reset") net->sigType="reset") when(ciIsNet(net "analog") net->sigType="analog"</code>
<i>None</i>	No action	<code>t</code>
<i>Alignment for Right Pins</i>	Generate Pin Alignment on the Right	<code>ciAlignPinsOnCellSide(cache instsNetsPins ?side "right")</code>
<i>Alignment for Bottom Pins</i>	Generate Pin Alignment on the Bottom	<code>ciAlignPinsOnCellSide(cache instsNetsPins ?side "bottom")</code>
<i>Alignment for Left Pins</i>	Generate Pin Alignment on the Left	<code>ciAlignPinsOnCellSide(cache instsNetsPins ?side "left")</code>
<i>Alignment for Top Pins</i>	Generate Pin Alignment on the Top	<code>ciAlignPinsOnCellSide(cache instsNetsPins ?side "top")</code>
<i>variantInfo for FETs</i>	Set variantInfo Property for FETs	<code>ciVariantInfoForFingersAndFingerWidth(instsNetsPins ?minFingerWidth "0.5u" ?maxFingerWidth "30u")</code>
<i>placerControlledWell</i>	Set placerControlledWell Property for pfets	<code>ciPlacerControlledWellGeneration(instsNetsPins)</code>
<i>Correlation</i>	Generate Correlation Constraint	<code>ciConCreateExpanded(cache `correlation ?members instsNetsPins ?params list(list("coefficient" 0.9)) ?note "CP: High Correlation for symmetric pair")</code>
<i>Matched Parameters for Same Cell Size and Bulk</i>	Generate Matched Parameters for Size and Value	<code>ciMatchedParamsForSameSizeInstances(cache instsNetsPins)</code>
<i>Cluster for Devices with common bulk</i>	Generate Cluster Constraint	<code>let ((cx) cx=ciConCreateExpanded(cache 'cluster ?members insts ?verbose nil ?note "CP: Cluster for devices with common bulk"))</code>
<i>Cluster for Standard Cells</i>	Generate Cluster Constraint	<code>let ((cx) cx=ciConCreateExpanded(cache 'cluster ?members insts ?verbose nil ?note "CP: Cluster for standard cells"))</code>
<i>Modgen for Same Cell Size and Bulk</i>	Generate Modgen Constraint	<code>ciModgenForSameCellSizeAndBulk(cache instsNetsPins)</code>

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

<i>Cluster Boundary for Cluster</i>	Generate Cluster Boundary Constraint	<code>ciClusterBoundaryForCluster(cache instsNetsPins)</code>
<i>Guard Ring for cluster</i>	Generate Guard Ring Constraint	<code>ciGuardRingForCluster(cache instsNetsPins)</code>
<i>Guard Ring for modgen</i>	Generate Guard Ring Constraint	<code>ciGuardRingForModgen(cache instsNetsPins)</code>
<i>Orientation for modgen</i>	Generate Orientation Constraint	<code>ciOrientationForModgen(cache instsNetsPins)</code>
<i>Alignment for Inverters</i>	Generate Alignment with vertAlign Axis	<code>instsNetsPins = setof(x instsNetsPins (cadr(x) == 'inst)) ciConCreateExpanded(cache `alignment ?members instsNetsPins ?axis "vertAlign" ?params list(list("side" "center")) ?note strcat("CP: Align NMOS and PMOS for an inverter"))</code>
<i>Distance for Inverters</i>	Generate Short Distance Constraint	<code>instsNetsPins = setof(x instsNetsPins (cadr(x) == 'inst)) ciConCreateExpanded(cache `distance ?members instsNetsPins ?params list(list("min" 2.0) list("max" 10.0)) ?note strcat("CP: Distance for the NMOS and PMOS of an inverter"))</code>
<i>Symmetry for Pins</i>	Generate Symmetry with vertSymm Axis	<code>let(() unless(ciAxisExists(cache "vertSymm") ciAxisCreate(cache "vertSymm")) ciConCreateExpanded(cache 'symmetry ?members setof(x instsNetsPins (cadr(x) == 'pin)) ?axis "vertSymm" ?verbose nil ?note strcat("CP: Symmetry for symmetrical pins"))</code>
<i>Symmetry for Nets</i>	Generate Symmetry with vertSymm Axis	<code>let(() unless(ciAxisExists(cache "vertSymm") ciAxisCreate(cache "vertSymm")) ciConCreateExpanded(cache `symmetry ?members instsNetsPins ?axis "vertSymm" ?verbose nil ?note "CP: Symmetry for pair of symmetric nets"))</code>
<i>Symmetry for Instance</i>	Generate Symmetry with vertSymm Axis	<code>let(() unless(ciAxisExists(cache "vertSymm") ciAxisCreate(cache "vertSymm")) ciConCreateExpanded(cache 'symmetry ?members instsNetsPins ?params list(list("mirror" 0)) ?axis "vertSymm" ?verbose nil ?note strcat("CP: Mirror Symmetry for symmetric instances"))</code>
<i>MatchedParameters for Symmetric Instances</i>	Generate Matched Parameters for Size and Value	<code>ciMatchedParamsForInstanceSymmetry(cache instsNetsPins)</code>
<i>MatchedParameters for Current Mirror</i>	Generate Matched Parameters for Size and Ratio	<code>ciMatchedParametersForCurrent_Mirror(cache instsNetsPins)</code>

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

<i>Cluster for Devices with same name and size</i>	Generate Cluster Constraint	<pre>let((cx) cx=ciConCreateExpanded(cache 'cluster ?members insts ?verbose nil ?note "CP: Cluster for devices with same name and size"))</pre>
<i>Cluster for Devices with same finger width</i>	Generate Cluster Constraint	<pre>let((cx) cx=ciConCreateExpanded(cache 'cluster ?members insts ?verbose nil ?note "CP: Cluster for devices with same finger width"))</pre>
<i>Modgen for Standard Cells</i>	Generate Modgen Constraint	<pre>ciConCreateExpanded(cache 'modgen ?members foreach(mapcar inst insts list(car(inst) cadr(inst) list(list("horiAlignment" "custom")))) ?verbose nil ?note "CP: modgen for standard cells")</pre>
<i>Distance + Matched Orientation + Vertical Alignment</i>	Generate Short Distance, Same Orientation and Alignment with vAlign Axis	<pre>unless(ciAxisExists(cache "vAlign") ciAxisCreate(cache "vAlign" list(list("direction" "vertical")))) ciConCreateExpanded(cache 'distance ?members insts ?params list(list("min" 2.0) list("max" 10.0)) ?note strcat("CP: Short distance between NMOS and PMOS of a transmission gate or inverter") ?verbose nil) && ciConCreateExpanded(cache 'relativeOrientation ?members insts ?note strcat("CP: Matched orientation of NMOS and PMOS of transmission gate or inverter") ?verbose nil) && ciConCreateExpanded(cache 'alignment ?members insts ?axis "vAlign" ?params list(list("side" "center")) ?note strcat("CP: Align vertically NMOS and PMOS of transmission gate or inverter") ?verbose nil)</pre>
<i>Common Bulk - Group + Matching Orientation</i>	Generate Cluster and Same Orientation Constraints	<pre>ciConCreateExpanded(cache 'cluster ?members insts ?note strcat("CP: Group together devices with common bulk") ?verbose nil) && when(cadr(instsNetsPins) ciConCreateExpanded(cache 'relativeOrientation ?members insts ?note strcat("CP: Matching orientation for devices with common bulk") ?verbose nil))</pre>
<i>Matched Parameters for Same Finger Width</i>	Generate Matched Parameters for Finger Width	<pre>ciMatchedFingerWidth(cache instsNetsPins)</pre>
<i>MOS Active Load - Matched + Symmetry + Distance</i>	Create MOS Active Load constraints	<pre>ciConCreate(cache `matchedParameters ?members list(car(instsNetsPins) cadr(instsNetsPins)) ?verbose nil) && ciConCreate(cache `symmetry ?members list(car(instsNetsPins) cadr(instsNetsPins)) ?verbose nil) && ciConCreate(cache `distance ?members list(car(instsNetsPins) cadr(instsNetsPins)) ?verbose nil)</pre>

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

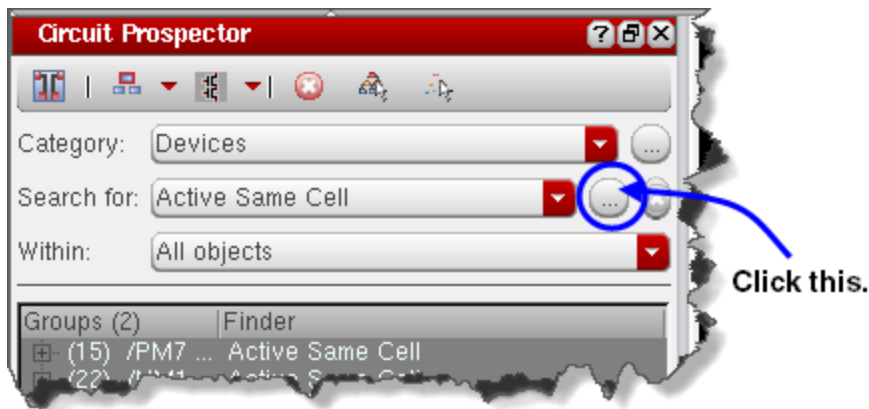
The above information can also be found in the [Edit Generator or Action Form](#) when editing a generator.

Creating Constraint Generators

Note: See also [Creating Custom Constraint Generators](#) in the *Virtuoso Unified Custom Constraints Configuration Guide*.

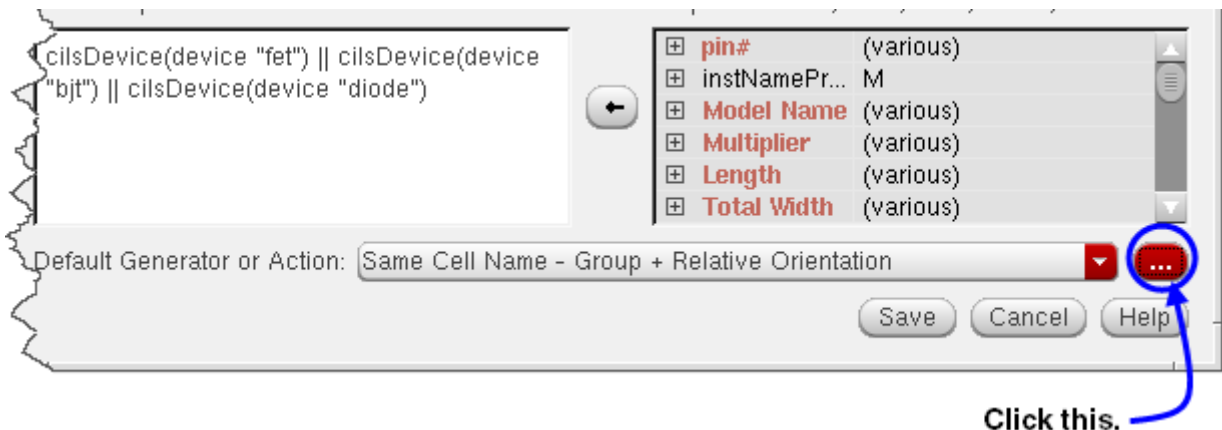
To create a new constraint generator:

1. Click the ... (ellipsis) button to the right of the *Search for* drop-down list box in the *Circuit Prospector* assistant.



This displays the [Edit Finder Form](#).

2. Click the ... (ellipsis) button to the right of the *Default Generator or Action* drop-down list box in the Edit Finder form.



Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

This displays the Edit Generator or Action Form.

The screenshot shows the 'Edit Generator or Action' window. The 'Generators or Actions:' list on the left includes items like 'Matched Parameters', 'Symmetry', 'vSymmetry', 'hSymmetry', 'Alignment', 'Orientation', 'Module', and the selected 'Same Cell Name - Group + Relative Orientation'. The 'Name' field on the right is populated with 'Same Cell Name - Group + Relative Orientation'. The 'Description' field contains 'Generate Cluster and Same Orientation Constraints'. The 'Expression' field contains the following code:

```
when( cadr(InstsNetsPins) ciConCreateExpanded(cache
'cluster ?members insts ?note strcat("CP: Group together
devices with same cell name") ?verbose nil)) &&
ciConCreateExpanded(cache 'relativeOrientation ?members
insts ?note strcat("CP: Matching orientation for devices with
same cell name") ?verbose nil)
```

3. Replace the name of the constraint generator in the *Name* field to create a new constraint generator.

Note: Editing the name of the current constraint generator will not remove it from the *Generators or Actions* list.

4. Edit the *Description* of the new constraint generator.
5. Edit the associated *Expression* for the new constraint generator.
6. Select the *Add* button to add the new constraint generator to the *Generators or Actions* list.
7. Repeat steps 3-6 as necessary to add more constraint generators.
8. Click the *Save* button to accept the constraint generator additions you made.

The Edit Finder form is displayed again and the new constraint generators are included in the *Default Generator or Action* drop-down list box.

Editing Constraint Generators

To edit a constraint generator:

1. Click the ... (ellipsis) button to the right of the *Search for* drop-down list box in the *Circuit Prospector* assistant.

This displays the Edit Finder Form.

2. Click the ... (ellipsis) button to the right of the *Default Generator or Action* drop-down list box in the Edit Finder form.

This displays the Edit Generator or Action Form.

3. Select the constraint generator you want to edit from the *Generators or Actions* list box.
4. Optionally, edit the *Description* of the constraint generator.
5. Optionally, edit the associated *Expression* for the constraint generator.
6. Repeat steps 3-5 as necessary to edit more constraint generators.
7. Click the *Save* button to accept the constraint generator edits you made.

The Edit Finder form is displayed again with the constraint generator edits applied.

Deleting Constraint Generators

To delete a constraint generator:

1. Click the ... (ellipsis) button to the right of the *Search for* drop-down list box in the *Circuit Prospector* assistant.

This displays the Edit Finder Form.

2. Click the ... (ellipsis) button to the right of the *Default Generator or Action* drop-down list box in the Edit Finder form.

This displays the Edit Generator or Action Form.

3. Select the constraint generator you want to delete from the *Generators or Actions* list box.
4. Click the *Delete* button.
5. Repeat steps 3-4 as necessary to delete more constraint generators.
6. Click the *Cancel* button to apply the constraint generator deletions and return to the Edit Finder form.

The constraint generators that you have deleted will no longer be listed in the *Default Generator or Action* drop-down list box.

Virtuoso Unified Custom Constraints User Guide

The Circuit Prospector Assistant

The Process Rule Editor

A **process rule** is a rule that constrains the implementation of a design based on the physical characteristics of the process technology being used. For example, a `minSpacing` rule specifies the minimum distance allowed between two shapes on a specified layer.

A **constraint group** is a collection of constraints and process rules created for a technology database or a specific design.

A process rule can exist only in a constraint group. Therefore, for a process rule to be applied to an object, design, or technology; it must be a member of a constraint group applied to that object, design, or technology.

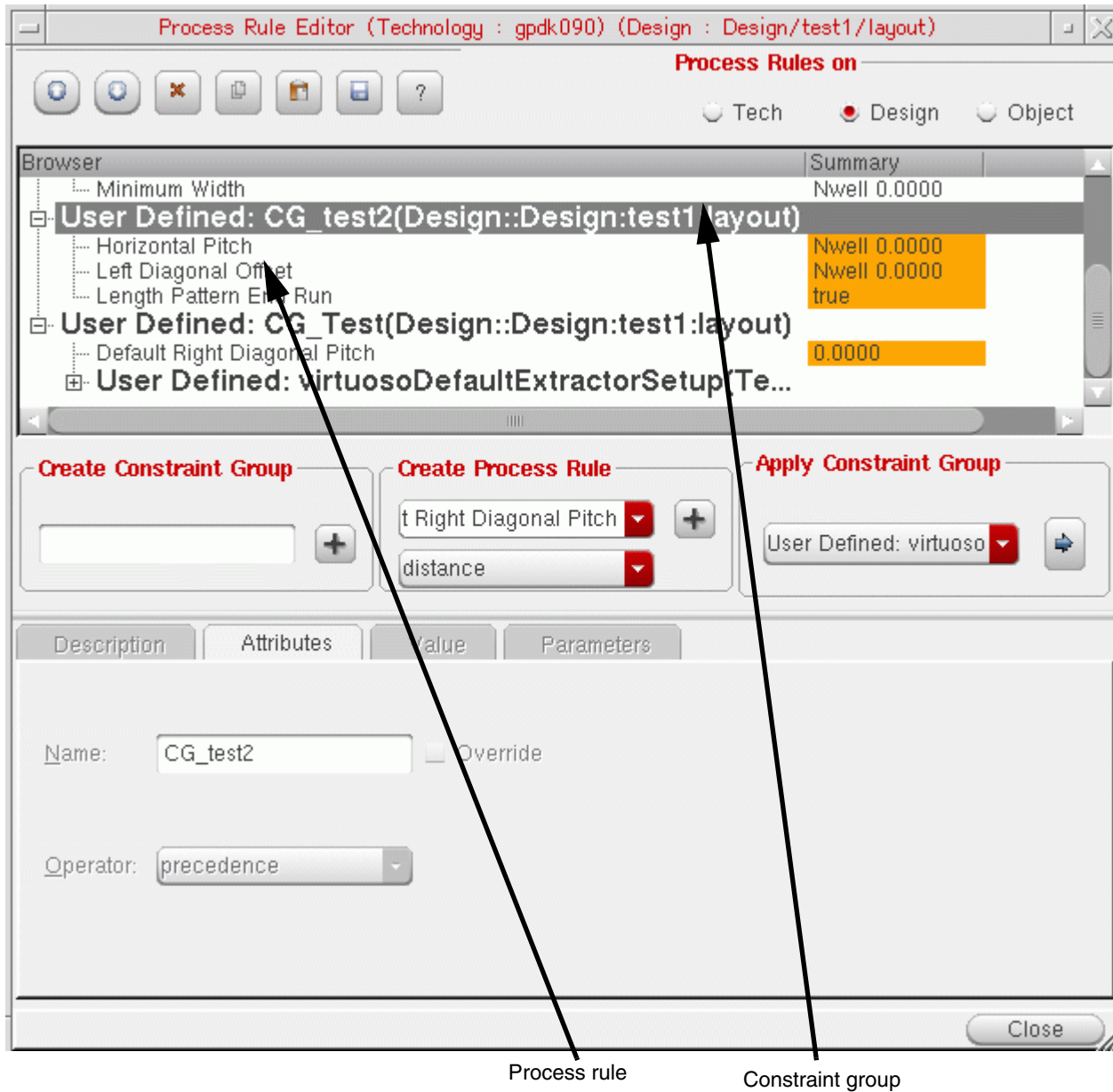
The Process Rule Editor lets you create new process rules and constraint groups and apply them to objects, constraints, designs, or technology databases; and change or remove existing process rules and constraint groups.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Important

If you want to quickly create and modify constraints or constraint groups that are useful for routing, use the Constraint Manager. If you want to create and modify any constraints or constraint groups to be applied to an object, design, or the technology you are using, use the Process Rule Editor.



The Process Rule Editor lets you create new process rules and constraint groups and apply them to objects, constraints, designs, or technology databases; and change or remove

existing process rules and constraint groups. The Process Rule Editor supports the creation of process rules and constraint groups for:

- Design objects such as *nets*
- Certain constraints such as *Diff Pair*, *Matched Length*, *Net Class*, *Shielding*, *Process Rule Overrides*, *Bus Net*, and *Symmetry*
- Entire design
- Technology databases

Important Points to Remember

- ❑ When dealing with Process Rule Override (PRO) on nets, you may encounter a mismatch between the representation on the schematic and the layout for the same PRO constraint. This may be due to an incorrect set up in the selected value for the Wire Editing constraint group (Layout: *Options – Editor* menu). The default value for this is the `virtuosoDefaultSetup` constraint group. To prevent the issue, ensure that the selected wire editing constraint group (default: `virtuosoDefaultSetup`) contains valid layers and valid vias.
- ❑ To use table-based spacing rules in a PRO constraint, specify the rules in a Constraint Group along with other rules in the design using the Process Rule Editor or the technology file. Then, choose this Constraint Group from the drop-down list for the *Default* parameter (that specifies the default constraint group for the member object) of the PRO constraint in the Constraint Manager. All of these can be done procedurally using SKILL as well. For more information, see [Creating a Constraint Group](#) and [Creating a New Process Rule](#).

For a complete list of process rules supported by the Process Rule Editor, see [Process Rules Supported by the Process Rule Editor](#).

Starting the Process Rule Editor

To start the Process Rule Editor:

- ➔ On the Constraint Manager toolbar, click the *Process Rule Editor* button ().

Note:

- ❑ Alternatively, right-click the design object in the layout canvas or the layout navigator and choose *Process Rule Editor* from the shortcut menu.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

- ❑ The Process Rule Editor command is available on the shortcut menu only for those objects that have a design issue and display a non-OK XL status.

The Process Rule Editor Graphical User Interface is displayed. The interface supports viewing multiple PRE windows simultaneously. Each PRE window responds to the changes in the design window from which it was opened.

For example, you have two designs open, `design_A` and `design_B`. Any changes you make to `design_A` will only update the Process Rule Editor you invoke from this design.

To make the Process Rule Editor respond to selections made in `design_B`:

- ➔ Open a new PRE window from the `design_B` Constraint Manager assistant.

Note: If the Process Rule Editor is already open for a design, a “re-open” will bring that editor up for viewing and make it available on top of the canvas.

Process Rules Supported by the Process Rule Editor

The table below lists the process rules and their corresponding Virtuoso constraint names. You can edit these rules by using the Process Rule Editor.

Process Rule Name	Virtuoso Constraint Name
<i>Accepted Tolerance</i>	<code>msTolerance</code>
<i>Access Point</i>	<code>accessPoint</code>
<i>Access to Pin</i>	<code>accessShift</code>
<i>Allowed Layer Geometry Orientation</i>	<code>layerShapeAngle</code> Techfile Name: <code>allowedShapeAngles</code>
<i>Allowed Length Ranges</i>	<code>allowedLengthRanges</code>
<i>Allowed Neighbor Width Range</i>	<code>allowedNeighborWidthRanges (Two layers)</code> <code>allowedNeighborWidthRanges (One layer)</code> <code>allowedNeighborWidthRangesOver</code>
<i>Allowed Shape Angles</i>	<code>allowedShapeAngles (0-layer)</code>
<i>Allowed Spacing Ranges Different Layers</i>	<code>allowedSpacingRanges (Two layers)</code>

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Process Rule Name	Virtuoso Constraint Name
<i>Allowed Spacing Ranges Same Layer</i>	<u>allowedSpacingRanges (One layer)</u>
<i>Allowed Width Ranges Same Layer</i>	<u>allowedWidthRanges</u>
<i>allowedCutClass</i>	<u>allowedCutClass</u>
<i>allowedGateLengthRanges</i>	<u>allowedGateLengthRanges</u>
<i>allowedGateWidthRanges</i>	<u>allowedGateWidthRanges</u>
<i>allowedPRBoundaryDimensions</i>	<u>allowedPRBoundaryDimensions</u>
<i>allowedSpanLengthRanges</i>	<u>allowedSpanLengthRanges</u>
<i>Cluster Distance</i>	<u>clusterDistance</u>
<i>Corner Fill Spacing</i>	<u>cornerFillSpacing</u>
<i>Cut Class</i>	cutClass Techfile Name: <u>cutClasses</u>
<i>Default Horizontal Routing Offset</i>	defaultHorizontalRoutingOffset Techfile Name: <u>horizontalOffset</u>
<i>Default Horizontal Routing Pitch</i>	defaultHorizontalRoutingPitch Techfile Name: <u>horizontalPitch (One layer)</u>
<i>Default Left Diagonal Offset</i>	defaultLeftDiagOffset Techfile Name: <u>leftDiagOffset</u>
<i>Default Left Diagonal Pitch</i>	defaultLeftDiagPitch Techfile Name: <u>leftDiagPitch (0-layer)</u>
<i>Default Right Diagonal Offset</i>	defaultRightDiagOffset Techfile Name: <u>rightDiagOffset (0-layer)</u>
<i>Default Right Diagonal Pitch</i>	defaultRightDiagPitch Techfile Name: <u>rightDiagPitch</u>
<i>Default Vertical Routing Offset</i>	defaultVerticalRoutingOffset Techfile Name: <u>verticalOffset</u>

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Process Rule Name	Virtuoso Constraint Name
<i>Default Vertical Routing Pitch</i>	defaultVerticalRoutingPitch Techfile Name: <u>verticalPitch (One layer)</u>
<i>Diagonal Shapes Allowed</i>	<u>diagonalShapesAllowed</u>
<i>Dummy Extension</i>	<u>dummyExtension</u>
<i>edgeMustCoincide</i>	<u>edgeMustCoincide</u>
<i>edgeMustOverlap</i>	<u>edgeMustOverlap</u>
<i>Error on Derived Layer</i>	<u>errorLayer</u>
<i>forbiddenCutClassSpacingRange</i>	<u>forbiddenCutClassSpacingRange</u>
<i>forbiddenEdgePitchRange</i>	<u>forbiddenEdgePitchRange</u>
<i>forbiddenProximitySpacing</i>	<u>forbiddenProximitySpacing</u>
<i>Gate Orientation</i>	<u>gateOrientation</u>
<i>Gate Spacing Ranges</i>	<u>gateSpacingRanges</u>
<i>Generic Constraint</i>	genericConstraint
<i>Horizontal Offset</i>	horizontalOffset Techfile Name: <u>horizontalOffset</u>
<i>Horizontal Pitch</i>	horizontalPitch Techfile Name: <u>horizontalPitch</u>
<i>Horizontal Placement Offset</i>	horizontalPlacementOffset Techfile Name: <u>horizontalOffset</u>
<i>Horizontal Placement Pitch</i>	horizontalPlacementPitch Techfile Name: <u>horizontalPitch</u>
<i>Illegal H Gate Pattern</i>	<u>illegalHGatePattern</u>
<i>Keep PRBoundary Shared Edges</i>	<u>keepPRBoundarySharedEdges</u>
<i>Keep Shared Edges</i>	<u>keepSharedEdges</u>
<i>Ladder Frequency</i>	<u>ladderFrequency</u>
<i>Ladder Offset</i>	<u>ladderOffset</u>

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Process Rule Name	Virtuoso Constraint Name
<i>Ladder Style</i>	<u>ladderStyle</u>
<i>Large Rectangular Via Array Allowed</i>	<u>largeRectViaArrayAllowed</u>
<i>Left Diagonal Offset</i>	leftDiagOffset Techfile Name: <u>leftDiagOffset</u> (1-layer)
<i>Left Diagonal Pitch</i>	leftDiagPitch Techfile Name: <u>leftDiagPitch</u> (1-layer)
<i>Length Pattern Accordion</i>	lengthPatternAccordion
<i>Length Pattern End Run</i>	lengthPatternEndRun
<i>Length Pattern RW Accordion</i>	lengthPatternRWAccordion
<i>Length Pattern Trombone</i>	lengthPatternTrombone
<i>lengthPatternOff</i>	lengthPatternOff
<i>Match Per Layer</i>	matchPerLayer
<i>Match Tolerance</i>	matchTolerance
<i>maxDiffusionLength</i>	<u>maxDiffusionLength</u>
<i>Maximum Coupling Diff</i>	maxCouplingDiff
<i>Maximum Coupling Diff Percent</i>	maxCouplingDiffPercent
<i>Maximum Current Density</i>	<u>maxACCurrentDensity</u>
<i>Maximum Density</i>	<u>maxDensity</u>
<i>Maximum Diagonal Edge Length</i>	<u>maxDiagonalEdgeLength</u>
<i>Maximum Different Density</i>	<u>maxDiffDensity</u>
<i>Maximum Extension</i>	<u>maxExtension</u>
<i>Maximum Floating Area</i>	<u>maxFloatingArea</u>
<i>Maximum Ground Cap Diff</i>	maxGroundCapDiff
<i>Maximum Ground Cap Percent</i>	maxGroundCapDiffPercent
<i>Maximum Length</i>	<u>maxLength</u>
<i>Maximum Number of Minimum Edges</i>	<u>maxNumMinEdges</u>

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Process Rule Name	Virtuoso Constraint Name
<i>Maximum number of via cuts of a cluster</i>	<u>maxViaClusterNumCuts</u>
<i>Maximum Poly Length</i>	<u>maxPolyLength</u>
<i>Maximum Routing Distance</i>	<u>maxRoutingDistance</u>
<i>Maximum Single Coupling</i>	maxSingleCoupling
<i>Maximum Single Coupling Percent</i>	maxSingleCouplingPercent
<i>Maximum Spacing For Filling</i>	<u>maxFilling</u>
<i>Maximum Spacing Same Layer</i>	<u>maxSpacing</u>
<i>Maximum Tap Spacing</i>	<u>maxTapSpacing</u>
<i>Maximum Touching Direction Length</i>	<u>maxTouchingDirectionLength</u>
<i>Maximum Width</i>	<u>maxWidth</u>
<i>maxNumCorners</i>	<u>maxNumCorners</u>
<i>minCenterLineSpacing</i>	minCenterLineSpacing
	Techfile Name:
	<u>minCenterLineSpacing (Two layers)</u>
<i>minChamferLength</i>	<u>minChamferLength</u>
<i>minClusterSpacing</i>	<u>minClusterSpacing (One layer)</u>
	<u>minClusterSpacing (Two layers)</u>
<i>minCornerExtension</i>	<u>minCornerExtension</u>
<i>minCornerSpacing</i>	<u>minCornerSpacing (One layer)</u>
	<u>minCornerSpacing (Two layers)</u>
<i>minDirectionalOverlap</i>	<u>minDirectionalOverlap</u>
<i>minEndOfLineToNotchSpacing</i>	<u>minEndOfLineToNotchSpacing</u>
<i>minExtensionSpacing</i>	<u>minExtensionSpacing</u>
<i>minExtensionToCenterLine</i>	<u>minExtensionToCenterLine</u>
<i>minExtensionToCorner</i>	<u>minExtensionToCorner</u>
<i>Minimum Adjacent Via Spacing</i>	<u>viaSpacing</u>
<i>Minimum Area</i>	<u>minArea</u>

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Process Rule Name	Virtuoso Constraint Name
<i>Minimum Area With Exceptions</i>	<u>minAreaEdgeLength</u>
<i>Minimum Center To Center Spacing Same Layer</i>	<u>minCenterToCenterSpacing</u>
<i>Minimum Centerline Extension</i>	<u>minCenterlineExtension</u>
<i>Minimum Corner To Corner Width</i>	<u>minCornerToCornerDistance</u>
<i>Minimum Coupling</i>	minCoupling
<i>Minimum Coupling Percent</i>	minCouplingPercent
<i>Minimum Cut Class Spacing - Different Layers</i>	minCutClassSpacing Techfile Name: <u>minCutClassSpacing (Two layers)</u>
<i>Minimum Cut Class Spacing - Same Layer</i>	minCutClassSpacing Techfile Name: <u>minCutClassSpacing (One layer)</u>
<i>Minimum Cut Routing Clearance</i>	minCutRoutingSpacing Techfile Name: <u>minCutRoutingSpacing (Two layers)</u>
<i>Minimum Density</i>	<u>minDensity</u>
<i>Minimum Diagonal Edge Length</i>	<u>minDiagonalEdgeLength</u>
<i>Minimum Diagonal Spacing Same Layer</i>	<u>minDiagonalSpacing</u>
<i>Minimum Diagonal Width</i>	<u>minDiagonalWidth</u>
<i>Minimum Differential Potential Net Spacing</i>	<u>minDiffNetSpacing</u>
<i>Minimum Distance from The PRBoundary to Enclosed Shape</i>	<u>minPRBoundaryInteriorHalo</u>
<i>Minimum Distance from The PRBoundary to Outside Shape</i>	<u>minPRBoundaryExteriorHalo</u>
<i>Minimum Dual End of Line Extension</i>	<u>minOppEndOfLineExtension</u>
<i>Minimum Edge Adjacent Distance</i>	<u>minEdgeAdjacentDistance</u>
<i>Minimum Edge Adjacent Length</i>	<u>minEdgeAdjacentLength</u>
<i>Minimum Enclosure Edge</i>	<u>minExtensionEdge</u>

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Process Rule Name	Virtuoso Constraint Name
<i>Minimum End of Line Adjacent To Step</i>	<u>minEndOfLineAdjacentToStep</u>
<i>Minimum End of Line Cut Spacing</i>	<u>minEndOfLineCutSpacing</u>
<i>Minimum End Of Line Edge Extension</i>	<u>minEndOfLineEdgeExtension</u>
<i>Minimum End Of Line Extension</i>	<u>minEndOfLineExtension</u>
<i>Minimum End Of Line Extension Spacing</i>	<u>minEndOfLineExtensionSpacing</u>
<i>Minimum End of Line Keepout</i>	<u>endOfLineKeepout</u>
<i>Minimum End Of Line Perpendicular Spacing</i>	<u>minEndOfLinePerpSpacing</u>
<i>Minimum End Of Line Spacing</i>	<u>minEndOfLineSpacing</u>
<i>Minimum End of Line to Corner Spacing</i>	<u>minEndOfLineToConcaveCornerSpacing</u>
<i>Minimum Extension Distance</i>	<u>minExtensionDistance</u>
<i>Minimum Extension on Long Side</i>	<u>minExtensionOnLongSide</u>
<i>Minimum Fill to Fill Spacing</i>	<u>minFillToFillSpacing</u>
<i>Minimum Filler To Shape Spacing, Same Layer</i>	<u>minFillToShapeSpacing</u>
<i>Minimum Gate Extension</i>	<u>minGateExtension</u>
<i>Minimum Hole Area</i>	<u>minHoleArea</u>
<i>Minimum Hole Width</i>	<u>minHoleWidth</u>
<i>Minimum Inner Vertex Clearance</i>	minInnerVertexSpacing Techfile Name: <u>minInnerVertexSpacing (Two layers)</u>
<i>Minimum Inner Vertex Spacing</i>	<u>minInnerVertexSpacing</u>
<i>Minimum Inside Corner Edge Length</i>	<u>minInsideCornerEdgeLength</u>
<i>Minimum Inside Corner Extension</i>	<u>minInsideCornerExtension</u>
<i>Minimum Inside Corner Overlap</i>	<u>minInsideCornerOverlap</u>
<i>Minimum Large Via Array Cut Spacing</i>	<u>minLargeViaArrayCutSpacing</u>

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Process Rule Name	Virtuoso Constraint Name
<i>Minimum Large Via Array Spacing</i>	<u>minLargeViaArraySpacing</u>
<i>Minimum Large Via Array Width</i>	<u>minLargeViaArrayWidth</u>
<i>Minimum Length</i>	<u>minLength</u>
<i>Minimum Neighboring Via Spacing</i>	<u>minNeighborViaSpacing</u>
<i>Minimum Notch Spacing</i>	<u>minNotchSpacing</u>
<i>Minimum Notch Span Spacing</i>	<u>minNotchSpanSpacing</u>
<i>Minimum Number of Cuts</i>	<u>minNumCut</u>
<i>Minimum OneDimension Array Structure</i>	<u>minOneDArrayStructure</u>
<i>Minimum Opposite Extension</i>	<u>minOppExtension</u>
<i>Minimum Opposite Span Spacing</i>	<u>minOppSpanSpacing</u>
<i>Minimum Orthogonal Via Spacing</i>	<u>minOrthogonalViaSpacing</u>
<i>Minimum Orthogonal Spacing Between Corners Same Layer</i>	<u>minConvexCornerSpacing</u>
<i>Minimum Outer Vertex Spacing</i>	<u>minOuterVertexSpacing</u>
<i>Minimum Outside Corner Edge Length</i>	<u>minOutsideCornerEdgeLength</u>
<i>Minimum Overlap Distance</i>	<u>minOverlapDistance</u>
<i>Minimum Overlap in Length Direction</i>	<u>minWireOverlap</u>
<i>Minimum Parallel Span Spacing</i>	<u>minParallelSpanSpacing</u>
<i>Minimum Parallel Via Spacing</i>	minParallelViaSpacing
	Techfile Name: <u>minParallelViaSpacing (One layer)</u>
<i>Minimum Parallel Via Spacing (Layer Array)</i>	minParallelViaSpacing
	Techfile Name: <u>minParallelViaSpacing (Two layers plus metal)</u>
<i>Minimum Parallel Within Via Spacing</i>	<u>minParallelWithinViaSpacing</u>
<i>Minimum Perimeter</i>	<u>minPerimeter</u>
<i>Minimum PRBoundary Extension</i>	<u>minPRBoundaryExtension</u>

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Process Rule Name	Virtuoso Constraint Name
<i>Minimum Protruded Spacing, Same Layer, Within Influencing Shape</i>	<u>minStubInfluenceSpacing</u>
<i>Minimum Protrusion Number of Cuts</i>	<u>minProtrusionNumCut</u>
<i>Minimum Protrusion Spacing</i>	<u>minProtrusionSpacing</u>
<i>Minimum Protrusion Width</i>	<u>protrusionWidth</u>
<i>Minimum Rectangular Area</i>	<u>minRectArea</u>
<i>Minimum Redundant Via Setback</i>	<u>redundantViaSetback</u>
<i>Minimum Same Metal Share Edge Via Spacing</i>	<u>minSameMetalSharedEdgeViaSpacing</u>
<i>Minimum Same Net Clearance</i>	minSameNetSpacing Techfile Name: <u>minSameNetSpacing (Two layers)</u>
<i>Minimum Same Net Spacing</i>	minSameNetSpacing Techfile Name: <u>minSameNetSpacing (One layer)</u>
<i>Minimum Side Extension</i>	<u>minSideExtension</u>
<i>Minimum Size</i>	<u>minSize</u>
<i>Minimum Spacing Between Notch And Shapes</i>	<u>minEndOfNotchSpacing</u>
<i>Minimum Spacing Different Layers</i>	minSpacing Techfile Name: <u>minSpacing (Two layers)</u>
<i>Minimum Spacing Over Layer</i>	<u>minSpacingOver</u>
<i>Minimum Spacing Same Layer</i>	minSpacing Techfile Name: <u>minSpacing (One layer)</u>
<i>Minimum Spacing to Short Edges</i>	<u>minEdgeLengthSpacing</u>
<i>Minimum Spacing, Same Layer, Within Influencing Shape</i>	<u>minInfluenceSpacing</u>
<i>Minimum Step Edge Length</i>	<u>minStepEdgeLength</u>

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Process Rule Name	Virtuoso Constraint Name
<i>Minimum Touching Directional Extension</i>	<u>minTouchingDirEnclosure</u>
<i>Minimum Touching Directional Spacing</i>	<u>minTouchingDirSpacing</u>
<i>Minimum Via Clearance</i>	minViaSpacing Techfile Name: <u>minViaSpacing (Two layers)</u>
<i>Minimum Via Extension</i>	<u>minViaExtension</u>
<i>Minimum Via Joint Extension</i>	<u>minViaJointExtension</u>
<i>Minimum Via Spacing</i>	minViaSpacing Techfile Name: <u>minViaSpacing (One layer)</u>
<i>Minimum Voltage Clearance</i>	minVoltageSpacing Techfile Name: <u>minVoltageSpacing (Two layers)</u>
<i>Minimum Voltage Spacing</i>	minVoltageSpacing Techfile Name: <u>minVoltageSpacing (One layer)</u>
<i>Minimum Width</i>	<u>minWidth</u>
<i>Minimum Wire Extension</i>	<u>minWireExtension</u>
<i>minInnerVertexProximityExtension</i>	<u>minInnerVertexProximityExtension</u>
<i>minJointCornerSpacing</i>	<u>minJointCornerSpacing</u>
<i>minNeighborExtension</i>	<u>minNeighborExtension</u>
<i>minNeighboringShapesSpacing</i>	<u>minNeighboringShapesSpacing</u>
<i>minProtrusionWidthLength</i>	<u>minProtrusionWidthLength</u>
<i>minQuadrupleExtension</i>	<u>minQuadrupleExtension</u>
<i>minSideSpacing</i>	<u>minSideSpacing (One layer)</u> <u>minSideSpacing (Two layers)</u>
<i>minSpanLengthSpacing</i>	<u>minSpanLengthSpacing</u>
<i>minStitchOverlap</i>	<u>minStitchOverlap</u>

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Process Rule Name	Virtuoso Constraint Name
<i>Mixed Signal Shield Style</i>	msShieldStyle
<i>msConnectSupplyDistance</i>	msConnectSupplyDistance
<i>msMaxCap</i>	msMaxCap
<i>msMaxRes</i>	msMaxRes
<i>msMinCap</i>	msMinCap
<i>msMinRes</i>	msMinRes
<i>Number of Strands</i>	<u>numStrands</u>
<i>Pin Cover</i>	pinCover
<i>Pin Staple</i>	pinStaple
<i>Rectangular Shape Orientation</i>	<u>rectShapeDir</u>
<i>Right Diagonal Offset</i>	rightDiagOffset
	Techfile Name: <u>rightDiagOffset</u> (One layer)
<i>Right Diagonal Pitch</i>	rightDiagPitch
	Techfile Name: <u>rightDiagPitch</u> (One layer)
<i>Route Maximum Length</i>	routeMaxLength
<i>Route Minimum Length</i>	routeMinLength
<i>Routing Directions</i>	<u>routingDirections</u>
<i>Same mask</i>	msSameMask
<i>Shape Required Between Spacing</i>	<u>shapeRequiredBetweenSpacing</u>
<i>shapeRequiredSpacing</i>	<u>shapeRequiredSpacing</u>
<i>shareShields</i>	shareShields
<i>Shield Gap</i>	shieldGap
<i>Shield Width</i>	shieldWidth
<i>snapGridHorizontal</i>	<u>snapGridHorizontal</u>
<i>snapGridVertical</i>	<u>snapGridVertical</u>
<i>Strand Spacing</i>	<u>strandSpacing</u>

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Process Rule Name	Virtuoso Constraint Name
<i>Strand Width</i>	<u>strandWidth</u>
<i>Tandem Layer Above</i>	tandemLayerAbove
<i>Tandem Layer Below</i>	tandemLayerBelow
<i>Tandem Width</i>	tandemWidth
<i>Taper Halo</i>	<u>taperHalo</u>
<i>Track Pattern</i>	trackPattern
<i>Trunk Accessing Num Cuts</i>	trunkAccessingNumCuts
<i>Use Non Orthogonal Routing Rules</i>	useNonOrthogonalRoutingRules
<i>Valid Layers</i>	<u>validLayers</u>
<i>Valid LPPs</i>	<u>validLayers</u>
<i>Valid Vias**</i>	<u>validVias</u>
<i>Vertical Offset</i>	<u>verticalOffset</u>
<i>Vertical Pitch</i>	verticalPitch
	Techfile Name: <u>verticalPitch (One layer)</u>
<i>Vertical Placement Offset</i>	verticalPlacementOffset
	Techfile Name: <u>verticalOffset</u>
<i>Vertical Placement Pitch</i>	verticalPlacementPitch
	Techfile Name: <u>verticalPitch</u>
<i>Via Control</i>	viaControl
<i>Via Orient Shift</i>	viaOrientShift
<i>Via Percentage</i>	viaPercent
<i>Via Stacking Allowed</i>	<u>stackable</u>
<i>Via Stacking Limits</i>	<u>viaStackingLimits</u>
<i>viaEdgeType</i>	<u>viaEdgeType</u>
<i>viaRequiredWithin</i>	<u>viaRequiredWithin</u>
<i>Wire Percentage</i>	wirePercent

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Note: All the other process rules are unsupported.

Important

You can delete unsupported process rules in the browser pane. But, you cannot create a new rule of an unsupported type; nor can you add or update the values for an existing rule of an unsupported type.

Note: **The Process Rule Editor allows use of via variants if they are already defined in the Technology or Design scope. The via variants are parameterized vias that can only be defined in the Technology and Design scopes. You can use these via variants in addition to the Standard and Custom vias.

Process Rule Editor Graphical User Interface

This section describes the seven different elements that comprise the Process Rule Editor graphical user interface. These GUI elements are:

- Toolbar
- Scope Selection Section
- Browser Pane
- Create Constraint Group Section
- Create Process Rule Section
- Dynamic Widget
- Apply Constraint Group Section

Before you explore the GUI elements, it will help to familiarize yourself with some useful features supported by the GUI:

- ☐ Resizable form that allows controlling the various widget sizes independently, helps reduce the widget area to a minimal size.
- ☐ Resizable Tree window widget that helps adjusting the height of the Tree and the Dynamic widget by moving the edge between the two areas.
- ☐ Support for assigning purposes to layers for the selected constraint.
- ☐ *Attributes* and *Parameters* tabs to specify the attribute and parameter values for the selected process rule.
- ☐ Commands on the shortcut menu to expand and collapse the constraint tree in the browser pane.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Toolbar

The toolbar lets you access certain frequently-used functions.



For more information on the individual buttons on the toolbar, see the table below.

Icon	Button Name	Description
	<i>Move Up</i>	Increases the precedence of the selected constraint group or process rule by moving it up one position in the browser pane. Note: This function is not available for objects at the top level of the browser tree, comprising the Constraint Groups. For more information, see Setting the Precedence of a Constraint Group or Process Rule.
	<i>Move Down</i>	Decreases the precedence of the selected constraint group or process rule by moving it down one position in the browser pane. Note: This function is not available for objects at the top level of the browser tree, comprising the Constraint Groups. For more information, see Setting the Precedence of a Constraint Group or Process Rule.
	<i>Remove</i>	Removes the selected constraint group or process rule from the Process Rule Editor. For more information, see Removing a Constraint Group or Process Rule.
	<i>Copy Rules</i>	Copies the selected rules to the Process Rule Editor clipboard. For more information, see Copying an Existing Process Rule.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Icon	Button Name	Description
	<i>Paste Rules</i>	Pastes the process rules in the clipboard into the selected constraint group. For more information, see Copying an Existing Process Rule .
	<i>Save Technology</i>	Saves the changes made since the Process Rule Editor was opened or the last time the <i>Save Technology</i> button was pressed. For more information, see Saving Changes and Closing the Process Rule Editor .
	<i>Help</i>	Opens this document.

Scope Selection Section

The **Process Rules on** section lets you choose the scope for the changes you make in the Process Rule Editor. The section is also referred to as the **Scope Selection** section. In this section, choose:

Technology to define rules and overrides that apply to all the designs using the current technology databases.

Design to define rules and overrides that apply for the current design only.

Object to define rules and overrides for the selected objects or constraints in the design.

Browser Pane

The browser pane displays the current constraint groups, process rules, and a summary of the layers used for the selected process rule and the process rule values. The browser pane is divided into the following columns:

- The *Browser* column lists the constraint groups in all the technology databases applicable for the design. These are listed as the top level tree items and the constraint groups and process rules defined within the top level constraint groups appear at the appropriate sublevels in the tree.
- The *Summary* column displays a single concatenated string representing the layers used for the selected process rule and the current value of the constraint. For example, if the current layer and width values for a *Minimum Width* constraint are *oxide* and

Virtuoso Unified Custom Constraints User Guide

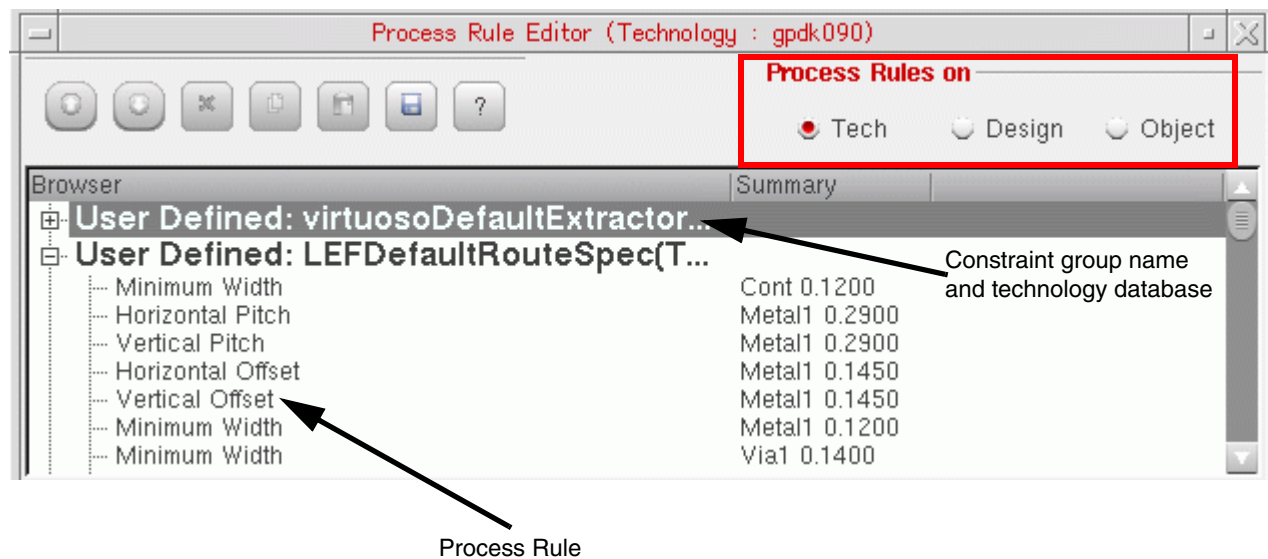
The Process Rule Editor

0.0250 respectively, the string displayed in the Summary column is oxide 0.0250. This is displayed in the figure below.



Note: If the *Summary* column is highlighted brown, it indicates that the process rule was created using the default values. If you modify the default values, the concatenated string displays the new values and the *Summary* column is not highlighted.

- The contents of the browser pane are determined by the option selected in the *Process Rules on* section, which is also called the *Scope Selection* section.



Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

- ❑ When set to *Technology*, the browser pane lists all the rules defined in the constraint groups across all the technology databases applicable to the current design.

The top level of the tree lists the name of the constraint group along with the technology database in which it resides. Expanding a constraint group reveals the rules defined within it. Provided the technology database is writable, you can override the setting for the current rules or create new rules in the selected constraint group.

- ❑ When set to *Design*, the browser pane lists the constraint groups and process rules defined only for the current design. Here too, provided the design is writable, you can override the setting for the current rules or create new rules that apply only to the current design.

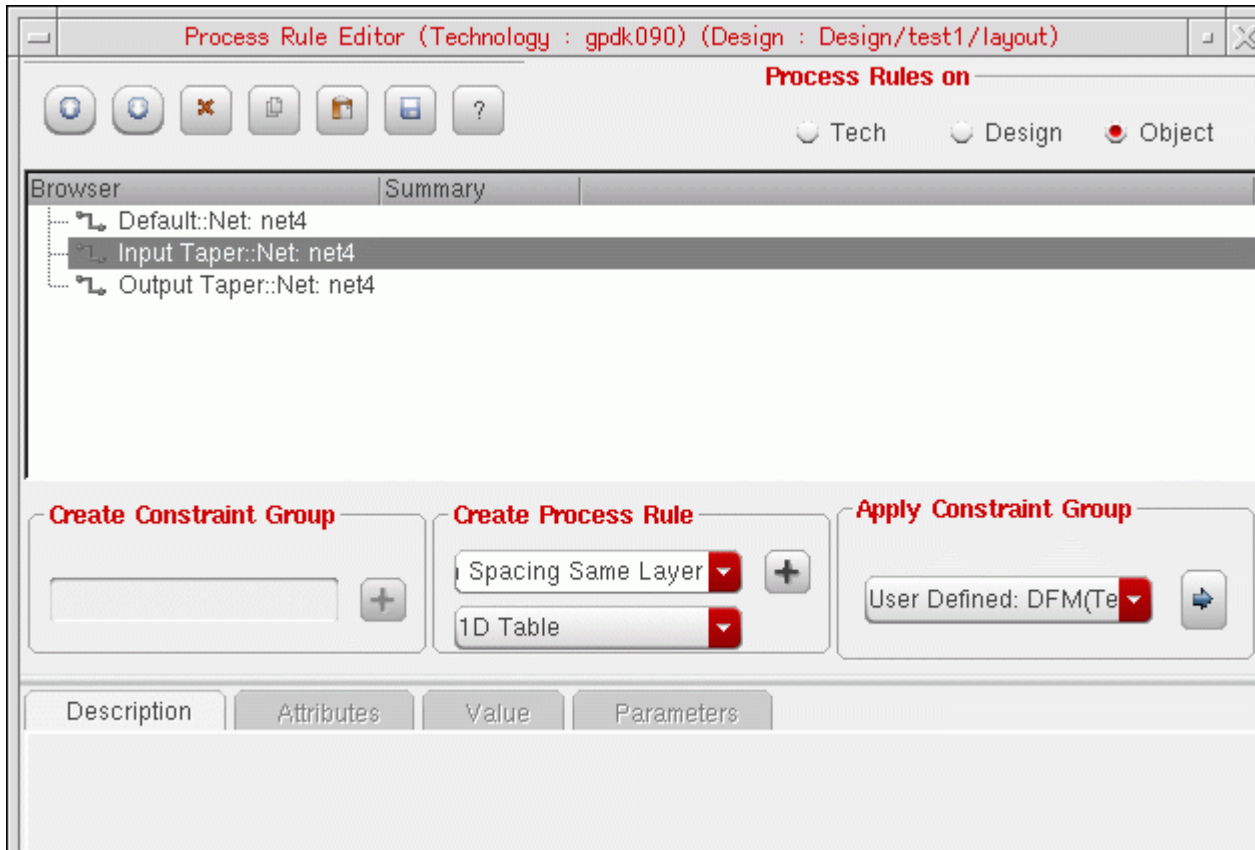


- ❑ When set to *Object*, the browser pane lists only the rules and constraint groups that apply for the selected object or constraint. For example, if you select a net, you see only the technology categories relevant for a net: *Default*, *Input Taper*, and *Output*

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Taper. In addition, you will see the rules associated with any constraints applied to the net (for example, a *Shielding* constraint).



Selecting one of these categories populates the **Create Process Rule** section with the rules you can define in that particular category. For more information, see [Creating a New Process Rule for a Selected Object](#).

Process Rule Editor Context Menu

The Process Rule Editor shortcut menu provides a number of context-specific options that you can apply on a constraint group or a process rule. To display the menu, right-click a constraint group or a process rule in the browser pane.

Note: Only those commands that are relevant to the selected item are active in the shortcut menu. The commands that are not applicable are inactive.

For example, when you are at the top-most level of the Constraint Group tree, the **Move Up** command appears inactive.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

The various commands in the context menu are tabulated below along with their description.

Command Name	Description
Copy Rules	Copies the selected process rules to the clipboard. This feature only copies the supported process rules. Any unsupported process rules are ignored by the command. For more information, see Copying an Existing Process Rule.
Paste Rules	Pastes the selected process rules from the clipboard to the specified location. You can paste the rules in the clipboard into as many locations as you want by repeatedly clicking the Paste Rules button. For more information, see Copying an Existing Process Rule.
Remove	Removes the selected constraint groups or process rules from the Process Rule Editor. If you delete a process rule in the browser in between copying and pasting, the deleted rule is also removed from the clipboard and is not subsequently pasted. For more information, see Removing a Constraint Group or Process Rule.
Move Up Move Down	Increases or decreases the precedence of the selected constraint group or process rule by moving it up or down one position in the browser pane. The commands are not available for the objects at the top level of the browser tree. For more information, see Setting the Precedence of a Constraint Group or Process Rule.

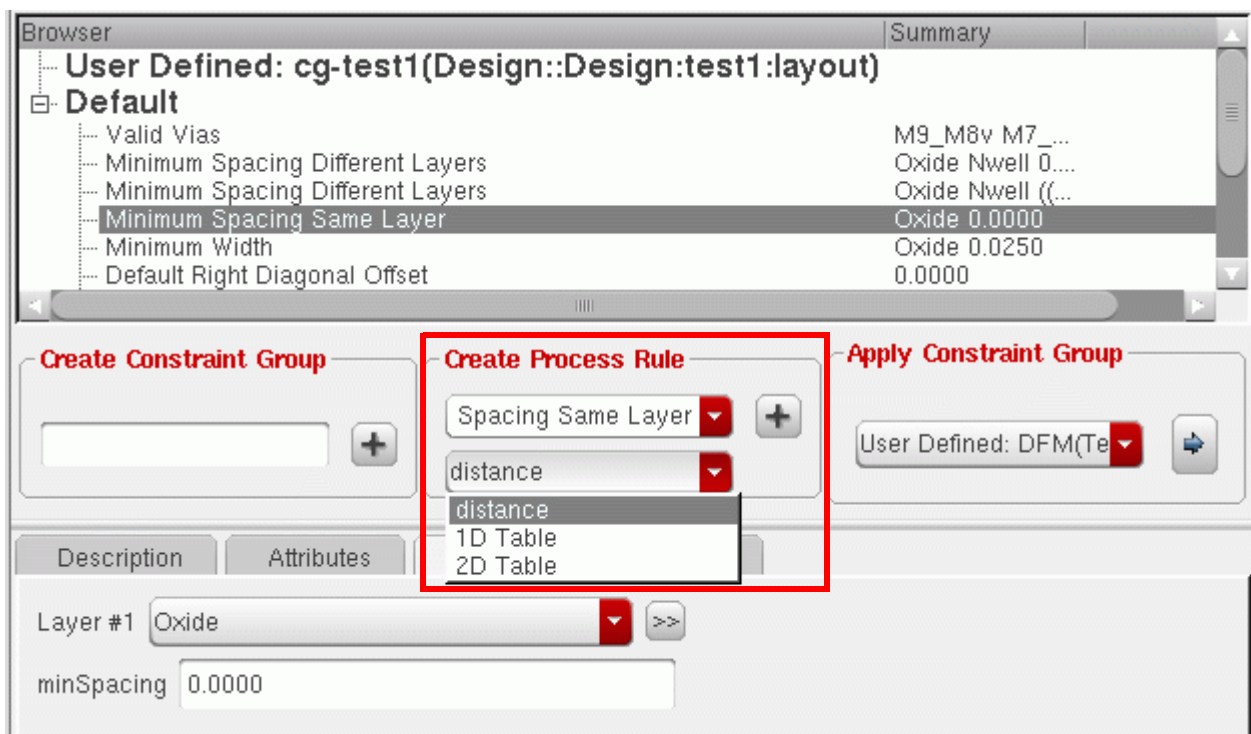
Create Constraint Group Section

You can use this section to create a new constraint group that contains process rules. For information about how to perform the task, see [Creating a Constraint Group](#).

Create Process Rule Section

You can use this section to create a new process rule in the selected constraint group. The **Create Process Rule** section includes:

- The drop-down list box at the top that enables you to specify the type of rule you want to create. The list contains only those rules that are valid for the selected constraint group or object.
- The drop-down list box at the bottom that enables you to choose an appropriate data value format for defining a particular rule. For example, you can define a *Minimum Spacing Same Layer* rule in terms of *distance* or as a *1D Table* or a *2D Table*, as displayed in the figure below.



The most common data value formats are:

- ❑ *distance*
- ❑ *1D table*
- ❑ *2D table*

Some other supported formats are:

- ❑ *2 layers, distance*

- ☐ *2 layers, 2D table*
- ☐ *2layers, dual distance*
- ☐ *dual distance table*
- ☐ *Layer Array*
- ☐ *Via Array*
- ☐ *direction*
- ☐ *BooleanValue*
- ☐ *voltage, spacing*
- ☐ *IntFltTblValue*
- ☐ *FltValue*

The type of process rule you choose from the drop-down list at the top governs the data value formats that are available in the Dynamic Widget.

- The **plus (+)** button that creates a new process rule and displays the process rule *Description and the Attributes, Value, and Parameters, if applicable.*

IntFltTblValue

This data format value represents a 1D integer lookup table value where the value is a float.

FltValue

This data format value represents a float value.

Dynamic Widget

You can use this widget to set the values for a new process rule. The widget comprises the following tabs:

- Description Tab
- Attributes Tab
- Value Tab
- Parameters Tab

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

You can also use these tabs to edit the values for an existing process rule. However, only relevant tabs are available for each rule.

The screenshot shows the Process Rule Editor with the 'Attributes' tab selected. The 'Name' field contains 'C__8' and the 'Hard' checkbox is checked. The 'ID' and 'Description' fields are empty. A 'Close' button is located at the bottom right.

Description Tab

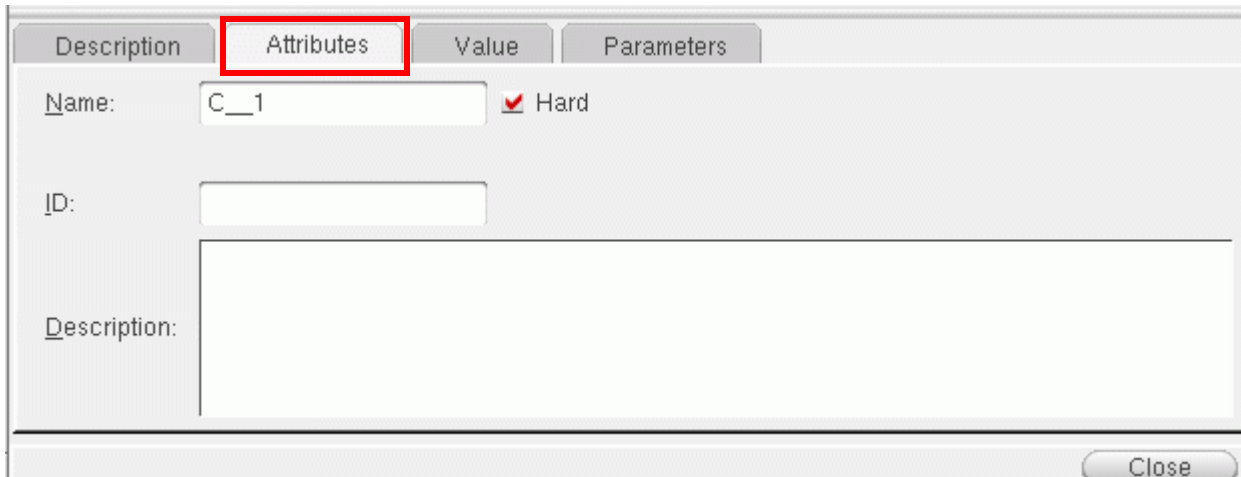
This tab provides a pictorial and a textual description of the selected process rule, as displayed in the figure below.

The screenshot shows the Process Rule Editor with the 'Description' tab selected. On the left, there is a diagram titled 'Metal Layers' showing two layers, L1 (red) and L2 (teal), with a spacing constraint 'C' between them. Below the diagram, it says 'Min Spacing'. On the right, there is a textual description: 'Specifies the minimum separation between two different shapes on two different layers. Spacing constraints apply only to edges that have no intersections. Spacings are specified from the outside edge of the shape on the first layer to the outside edge of the shape on the second layer. This constraint definition is symmetric, implying that the minimum spacing between layer1 and layer2 is the same as the minimum spacing between layer2 and layer1.' A 'Close' button is located at the bottom right.

The textual description is a generic definition of the process rule and the associated picture elaborates on this definition. The contents of the tab cannot be edited.

Attributes Tab

You can use this tab to specify the process rule attributes, such as name, ID, and description. In addition, you can use the tab to specify if the process constraint is hard or soft.



The screenshot shows the 'Attributes' tab selected in the 'Process Rule Editor'. The tab is highlighted with a red box. The form contains the following fields and controls:

- Name:** A text field containing 'C_1' and a checkbox labeled 'Hard' which is checked.
- ID:** An empty text field.
- Description:** A large empty text area.
- Close:** A button in the bottom right corner.

- **Name:** A unique, default name is displayed in this field for every process rule that you add. You can replace this name with a new name, if needed.
Note: The new name that you specify for a process rule must be unique. Otherwise, an error message is displayed in the CIW stating that the name could not be changed because a process rule by that name already exists.
- **Hard:** Select the *Hard* check box for a process rule if you do not want another process constraint to over-ride this rule.
- **ID:** Use the *ID* field to assign an ID to the process rule.
- **Description:** Use the *Description* field to provide a brief description of the process rule in the context of its current scope (*Tech. Design*, or *Object*) or the constraint group to which the rule is applied.

Note: Any values that you update on the *Attributes* tab are automatically saved for the current instance.

Value Tab

You can use this tab to select a layer such as `Metal2`, a constraint value such as `0.03`, and an appropriate layer purpose, such as `drawing`, for the process rule you are creating. You can use the layer-purpose widget provided by the *Value* tab to create layer-purpose pairs, as

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

required. For more information about creating layer-purpose pairs, see [Creating a Layer-Purpose Pair](#).

The screenshot shows the 'Value' tab of the Process Rule Editor. The 'Layer #1' dropdown is set to 'Metal2' and the 'drawing' dropdown is set to 'drawing'. The 'minCenterToCenterSpacing' field is set to '0.03'. The 'Add' button is highlighted in red.

Some process rules, such as *Minimum Spacing Different Layers*, may require you to select two layers and specify the associated purposes.

The screenshot shows the 'Value' tab of the Process Rule Editor. The 'Layer #1' dropdown is set to 'Oxide' and the 'Layer #2' dropdown is set to 'Poly'. The 'minSpacing' field is set to '0.0000'. The 'Add' button is highlighted in red.

Parameters Tab

You can use this tab to specify the parameters, such as *distance*, *width*, and *endOfLineSpace*, that define the conditions governing when or how a process rule can be applied.

Depending on the process rule you select, the parameters displayed on this tab vary.

- If a process rule has no associated parameters, the *Parameters* tab is unavailable.
- If a process rule has several associated parameters, default values are displayed for all the constraint parameters. You can modify these default values, as required.
- If a process rule has several parameters of which some are currently not supported, these parameters display “UNSUPPORTED” next to their name. The supported parameters display a default value that you can modify.

If you modify the default value of a parameter, the *Restore Default Value* () button appears next to the new parameter value. You can click this button to revert to the default value for this parameter.

- The value you enter for a parameter is validated against the data types allowed for that parameter, such as integer or string. An unsupported data type is not accepted.

Note: Any values that you update on the *Parameters* tab are automatically saved. Therefore, you do not need to click *Update*.

Apply Constraint Group Section

This section lets you apply a constraint group from the same or different scope to the selected constraint group in your current scope. In other words, the **Apply Constraint Group** section acts as a link, making all the applicable Constraint Groups available in the current scope.

For example, you can select a constraint group from the *Tech* scope and apply it to a custom constraint in the *Design* scope. However, you can apply a constraint group to a selected constraint only once. For information on how to perform the tasks, see [Applying a Constraint Group](#).

Note: The Constraint Groups you create in the *Tech* scope will also be available in the *Design* and *Object* scopes as these are applicable for all the designs that use the current process technology. However, the custom constraints you create in the *Design* scope will only be available in the *Design* and *Object* scopes as these are applicable at a design level and do not impact the overall process technology.

For more information about using the *Apply Constraint Group* section, see [Applying a Constraint Group](#).

For information on the tasks you can perform in the Process Rule Editor, see the topics listed below.

- [Creating a Constraint Group](#)
- [Customizing a Constraint Group](#)
- [Creating a New Process Rule](#)
 - [Creating a New Process Rule in the Technology or Design Mode](#)
 - [Creating a New Process Rule for a Selected Object](#)
 - [Creating a New Process Rule for a Constraint](#)
 - [Copying an Existing Process Rule](#)
 - [Creating a Layer-Purpose Pair](#)
 - [Applying Valid LPPs Process Rule](#)
 - [Creating a Voltage Dependent Process Rule](#)
 - [Editing Existing Process Rule Values](#)
- [Applying a Constraint Group](#)
- [Setting the Precedence of a Constraint Group or Process Rule](#)
- [Virtuoso Generated Constraint Groups](#)
- [Applying an Existing Constraint Group to a Selected Object](#)
- [Removing a Constraint Group or Process Rule](#)
- [Saving Changes and Closing the Process Rule Editor](#)

Creating a Constraint Group

You can create a new constraint group only in the *Technology* or *Design* mode by using the **Create Constraint Group** field. In the *Object* mode, the field is inactive.

Note: You can create an appropriate Virtuoso-generated constraint group or apply an existing constraint group to a selected object. For more information, see [Applying a Virtuoso Generated Constraint Group](#) and [Applying an Existing Constraint Group to a Selected Object](#).

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

To create a new constraint group:

1. On the Constraint Manager toolbar, click the *Process Rule Editor* button ().

Note: Alternatively, right-click the design object in the layout canvas or the layout navigator and choose *Process Rule Editor* from the shortcut menu.

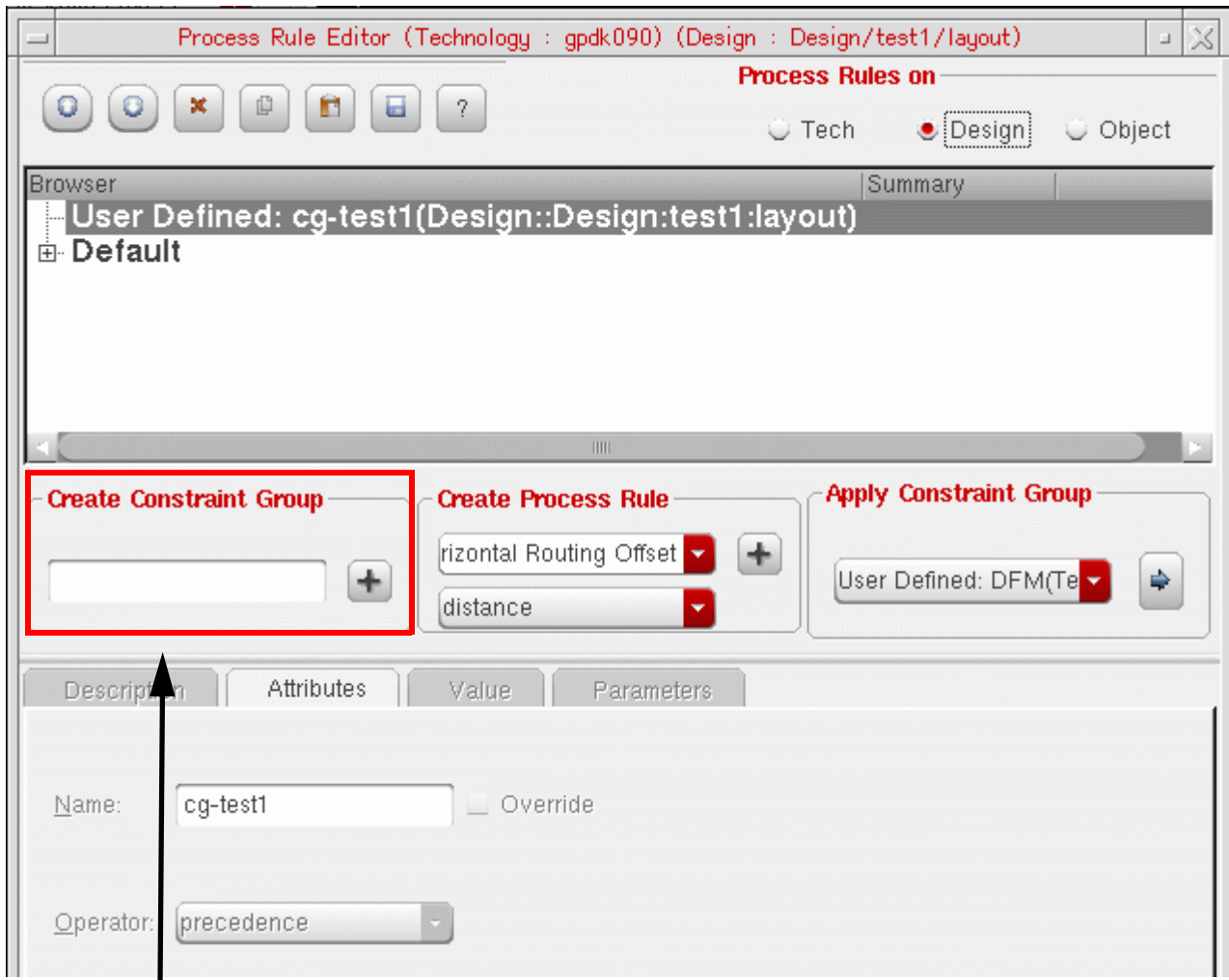
The Process Rule Editor Graphical User Interface is displayed.

2. In the *Process Rules on* section, choose one of the following:
 - ☐ *Technology* to create a constraint group applicable for all the designs that use the current process technology.
 - ☐ *Design* to create a constraint group applicable only for the current design.
 - ☐ *Object* to create a constraint group for the selected object or constraint in the design.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

3. Type a name for the new constraint group in the *Create Constraint Group* section.



Type a name for the new
constraint group

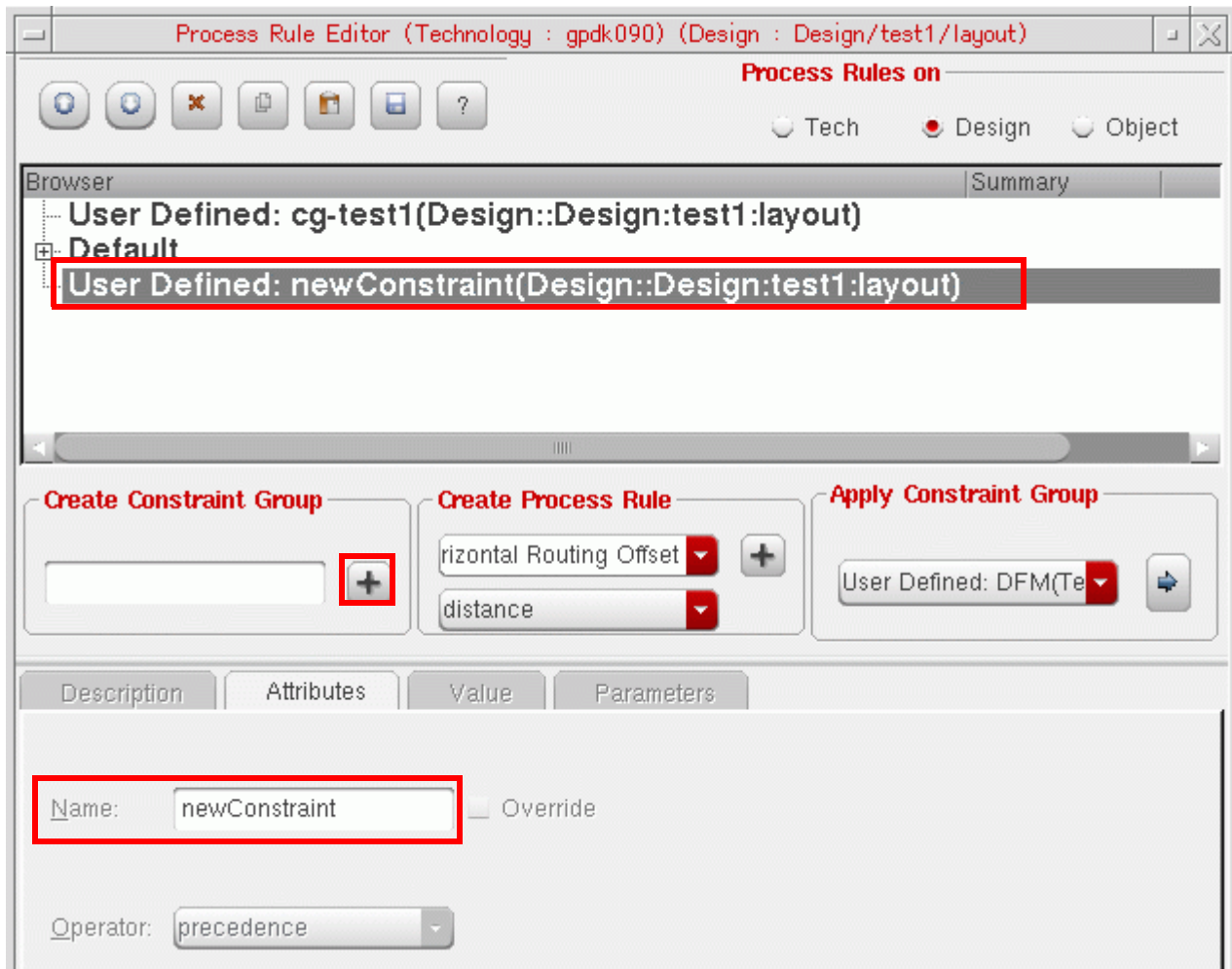
Note: You cannot prefix a constraint group name with an underscore (_). This convention is reserved for Cadence internal use.

4. Click the **plus (+)** button in the *Create Constraint Group* section to add the new group to the database you selected.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Note: Alternatively, press the **Enter** key.



The constraint group you created is added to the browser pane as displayed in the figure above.

Note: After the new constraint group is added to the browser pane, the *Name* field on the *Attributes* tab displays the name of the new constraint group.

The new constraint group is created as an empty *floating constraint group* for the selected mode (technology or design).

Customizing a Constraint Group

There can be constraints that are specified by adding information with the PRE. This information can be directly added or viewed in the constraint manager when you want to apply a single pre-existing constraint group to the available fields in the PRE. However, there are cases where you may not have any available constraint groups that meet your requirement. Therefore, you can generate customized constraint groups that are specific to your naming convention and as per the required design or style.

This customization can be provided in the form of the following SKILL functions:

```
isConstraintGroupApplicable(  
    ciCon, CGDefName, CG  
)  
=> t|nil  
  
listGroupsThatCanBeGenerated(  
    ciCon, CGDefName  
)  
=> a list of strings)  
  
generateConstraintGroup(  
    ciCon, CGDefName, CGName  
)  
=> a SKILL Constraint Group Object
```

where:

- **ciCon** is the SKILL object representing the current constraint in the constraint manager.
- **cgDefName** is the name of the CI constraint group's definition that you are trying to fill. For example, **shielding**, **transReflexive**, **reflexive**, **default**, **taper**, **inputTaper**, and **outputTaper**.
- **CG** is the SKILL object representing a constraint group that you want to test.
- **CGName** is the string name of the constraint group that you want to generate.

In addition, you can override the built-in functions by:

- Creating replacement functions with arguments and return values as shown above.
- Calling the [ciSetCMCGSKILLCallbacks](#) SKILL function.

You can use `nil` for any of these values if you want to reset them to default.

Creating a Custom Constraint Group and Defining Default Parameter Values

Alternatively, you can also create a custom constraint group and define default parameter values using the SKILL commands `ciAddRuleGroup` and `ciAddProcessRules` in the CIW. For example,

1. In the CIW, create a new constraint group called `myRG1`, using the `ciAddRuleGroup` command:

```
cv=geGetEditCellView()  
myRG1=ciAddRuleGroup(cv "myRG1")
```

2. Define default values for the top and bottom layers of the constraint group `myRG1` using the `ciAddProcessRules` command:

```
ciAddProcessRules(myRG1 '( ("minWidth" "Metal1" 1.0) ("minWidth" "Metal2" 1.2)  
("validLayers" nil ("Metal1" "Metal2" "Metal3"))))
```

The new constraint group can now be selected as the *Default Group (Constraint Group)* on the Constraint Parameter Editor.

Creating a New Process Rule

You can create new process rules for the entire technology or design, for one or more selected objects, or for a constraint applied to an object in your design. You can also copy process rules from one or more constraint groups to another constraint group. In addition, you can edit an existing process rule and specify layer purposes. You can also create and apply rules that are voltage dependent.

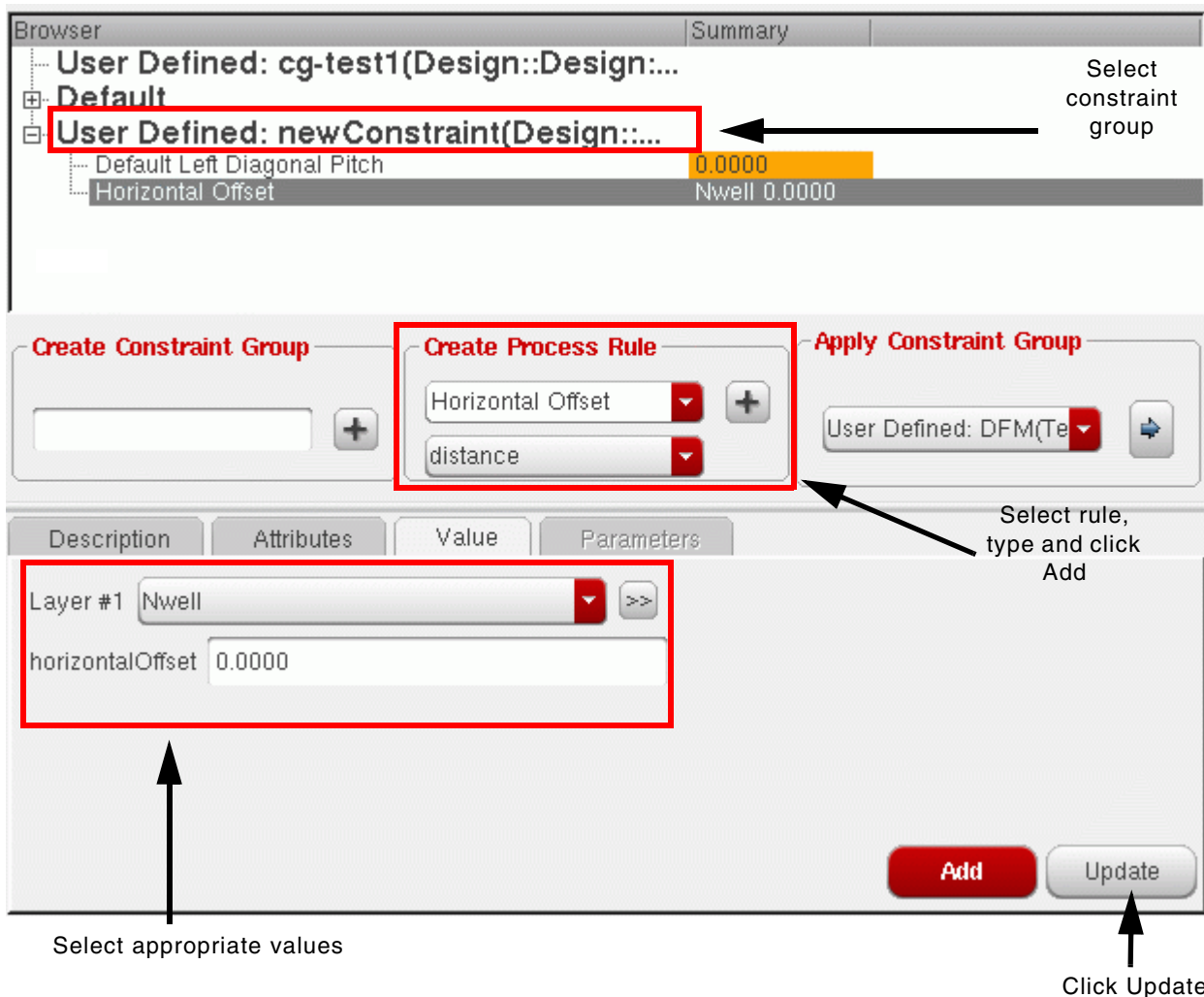
- Creating a New Process Rule in the Technology or Design Mode
- Creating a New Process Rule for a Selected Object
- Creating a New Process Rule for a Constraint
- Copying an Existing Process Rule
- Creating a Layer-Purpose Pair
- Applying Valid LPPs Process Rule
- Creating a Voltage Dependent Process Rule
- Editing Existing Process Rule Values

Creating a New Process Rule in the Technology or Design Mode

To create a new process rule in a constraint group using the *Technology* or *Design* mode:

1. In the browser pane, choose the constraint group in which you want to create the new rule.
2. In the **Create Process Rule** section, choose
 - ☐ The type of rule to create from the first drop-down list.
 - ☐ The value type, where appropriate, from the second drop-down list.

Note: Each list is seeded appropriately depending on what is selected in the browser pane.



3. Click the *plus (+)* button in the *Create Process Rule* section to add a new process rule to the selected constraint group.

Note: Alternatively, press the *Enter* key.

The process rule is added to the list. The *Summary* field in the browser pane is highlighted brown to indicate that the rule was created using the default values. You can modify the rule by changing the default values. This also removes the highlight from the *Summary* column.

4. Modify the rule by specifying appropriate data on the *Attributes*, *Value*, and *Parameters* tabs.

Note: Only relevant tabs are available for a process rule.

5. Click *Update* to apply the value you set in the *Value* tab.

Note: The changes you make on the *Attributes* and *Parameters* tabs are automatically saved. Therefore, you do not need to click the *Update* button to save these changes.

The new values that you specify on the *Value* tab appear in the *Summary* column of the browser pane.



Tip

To quickly add more process rules of the same type, click the *Add* button instead of *Update*. Each click adds a new rule with the same values displaying in the *Summary* column as the last one you added. The *Summary* column appears brown to indicate that the process rule values might need to be changed.

6. Select each new rule you added and edit the values, as required.

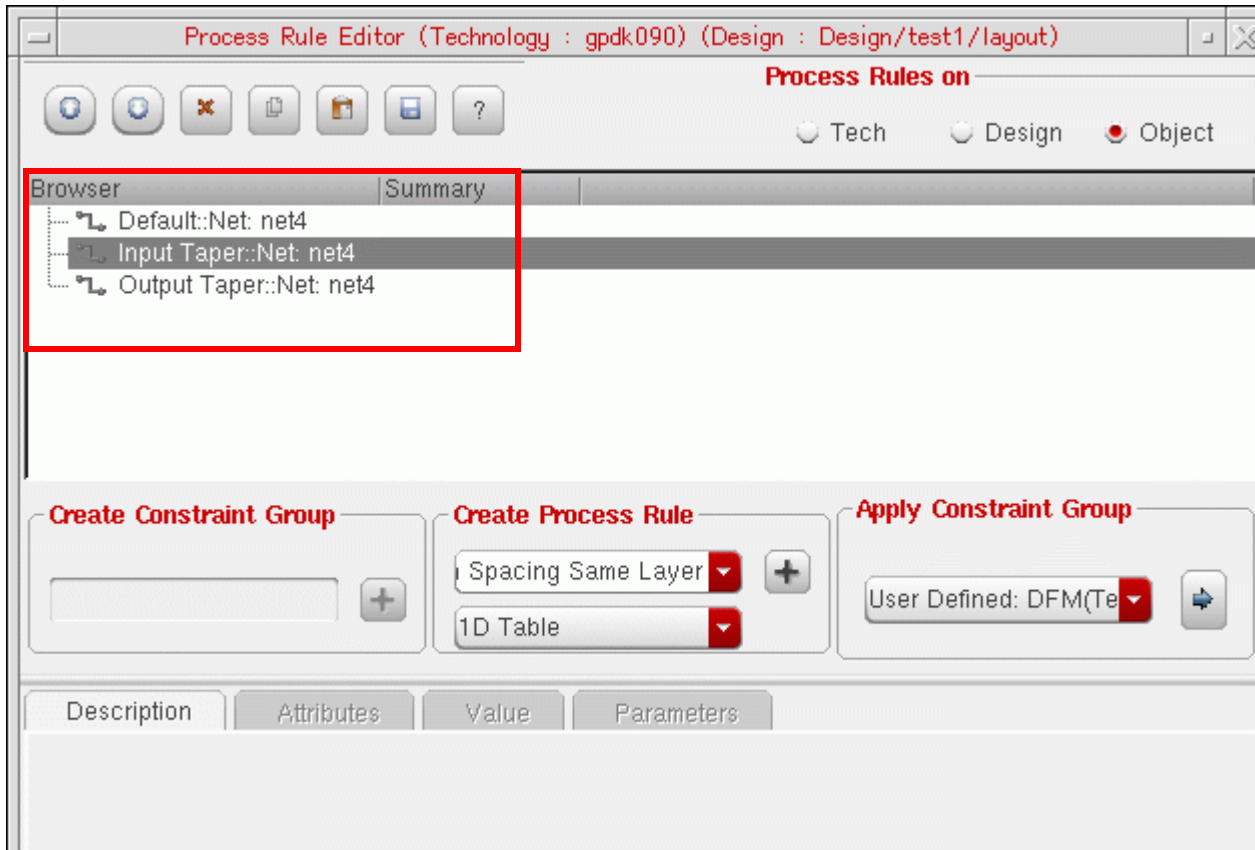
Creating a New Process Rule for a Selected Object

You can use the Process Rule Editor to create process rules for the selected objects in your design. In this mode, the browser pane shows only the categories that are applicable for the

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

selected objects. For example, if you select a net object, you see only the *Default*, *Input Taper*, and *Output Taper* categories.



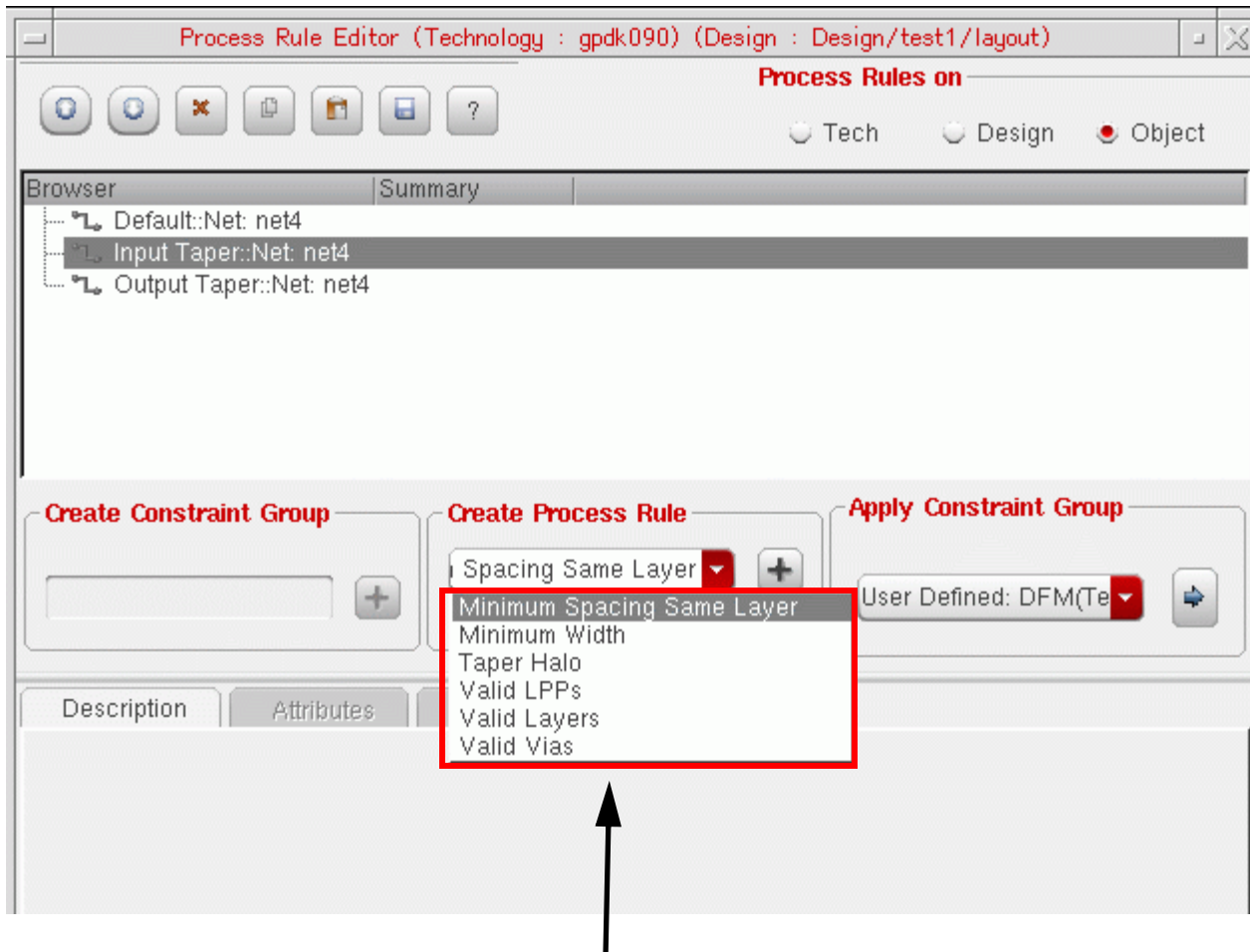
Selecting one of the categories populates the *Create Process Rule* section with the rules you can define in that particular category. For example, for *Input Taper*, as displayed in the figure above, you can define the following rules only:

- *Minimum Spacing Same Layer*
- *Minimum Width*
- *Taper Halo*
- *Valid LPPs*
- *Valid Layers*

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Valid Vias



Only valid rule types are
made available

Note: When adding process rules in the Object mode, the Process Rule Editor creates a Process Rule Overrides constraint group on the selected net(s).

To create a process rule in the *Object* mode:

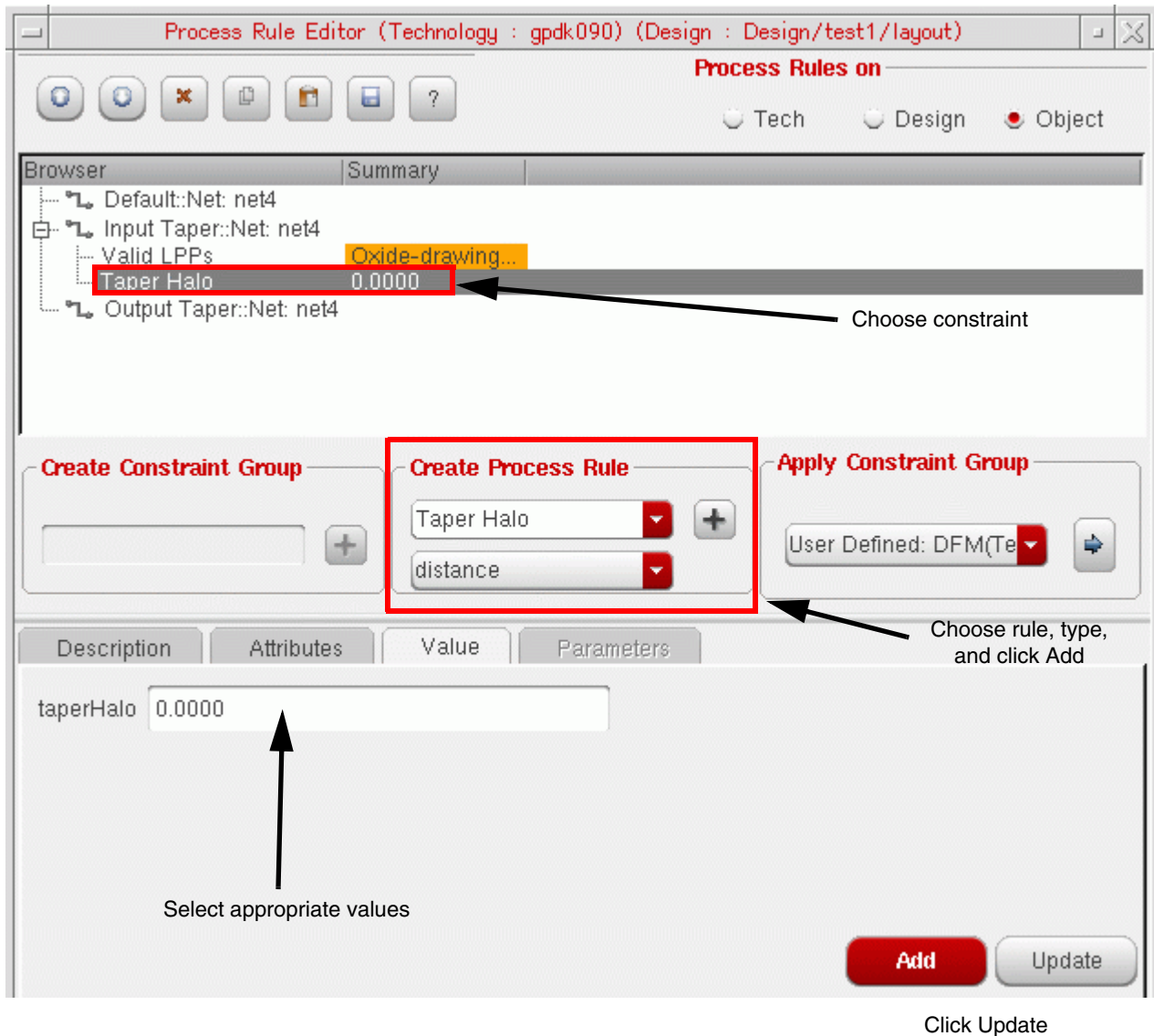
1. Use the Navigator assistant or the design window to select the object to which you want to add a process rule.
2. In the browser pane, choose the category for which you want to create the rule.
3. In the **Create Process Rule** section, choose
 - ❑ The type of process rule to create from the first drop-down list.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

- ❑ The value type, where appropriate, from the second drop-down list.

Note: Each list is seeded appropriately depending on what is selected in the browser pane.



4. Click the *plus (+)* button in the *Create Process Rule* section to add the new process rule to the selected constraint group.

Note: Alternatively, press the *Enter* key.

The process rule is added to the list. The *Summary* field in the browser pane is highlighted brown to indicate that the rule was created using the default values.

5. Modify the rule by changing the default data on the *Attributes*, *Value*, and *Parameters* tabs.

Note: Only relevant tabs are available for a process rule.

6. Click *Update* to apply the new values you set in the *Value* tab.

Note: The changes you make on the *Attributes* and *Parameters* tabs are automatically saved. Therefore, you do not need to click the *Update* button to save these changes.

The new values that you specify on the *Value* tab appear in the *Summary* column of the browser pane. The brown highlight disappears indicating that the rule has been modified.



To quickly add more process rules of the same type, click the *Add* button instead of *Update*. Each click adds a new rule with the same values displaying in the *Summary* column as the last one you added. The *Summary* column appears brown to indicate that the process rule values might need to be changed.

7. Select each new rule you added and edit the values, as required.

You can also create an appropriate Virtuoso-generated constraint group or apply an existing constraint group to a selected object. For more information, see [Applying a Virtuoso Generated Constraint Group](#) and [Applying an Existing Constraint Group to a Selected Object](#).

Creating a New Process Rule for a Constraint

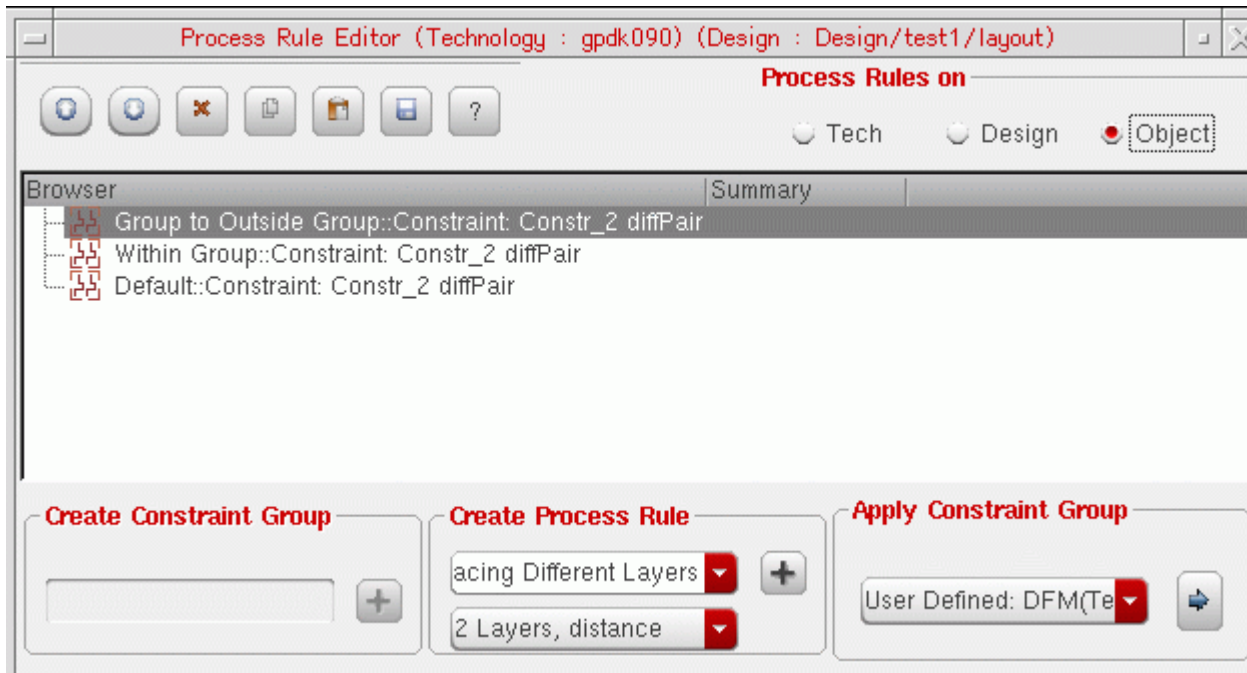
You can use the Process Rule Editor to create process rules for constraints selected in the Constraint Manager assistant. The following constraints are supported.

- *Diff Pair*
- *Matched Length*
- *Net Class*
- *Shielding*
- *Symmetry*
- *Bus Net*
- *Process Rule Overrides*

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

When you define process rules for constraints, the browser pane displays only the process rule categories that apply to the selected constraint. For example, for a *Diff Pair* constraint, you can see the *Group to Outside Group*, *Within Group*, and *Default* categories.



Note:

- *Group to Outside Group* refers to rules that apply between the members of the current constraint and the objects outside the current constraint.
- *Within Group* refers to rules that apply between the members of a constraint.
- *Default* refers to the default set of constraints that apply to the members of the current constraint.

Selecting one of these categories populates the **Create Process Rule** section with the rules you can define in that particular category. For example, for *Group to Outside Group*, you can define the *Minimum Spacing Different Layers* and *Minimum Spacing Same Layer* rules.

To create a process rule for a constraint;

1. Select the constraint for which the rule is to be created in the Constraint Manager assistant.
2. If the Process Rule Editor is not displayed, click the *Process Rule Editor* button () on the Constraint Manager toolbar.

Note: Alternatively, right-click the design object in the layout canvas or the layout navigator and choose *Process Rule Editor* from the shortcut menu.

3. In the Process Rule Editor browser pane, select the category for which you want to create the rule. For this example, let us select the *Group to Outside Group* category.

4. In the **Create Process Rule** section, choose

- ☐ The type of process rule to create from the first drop-down list.
- ☐ The value type, where appropriate, from the second drop-down list.

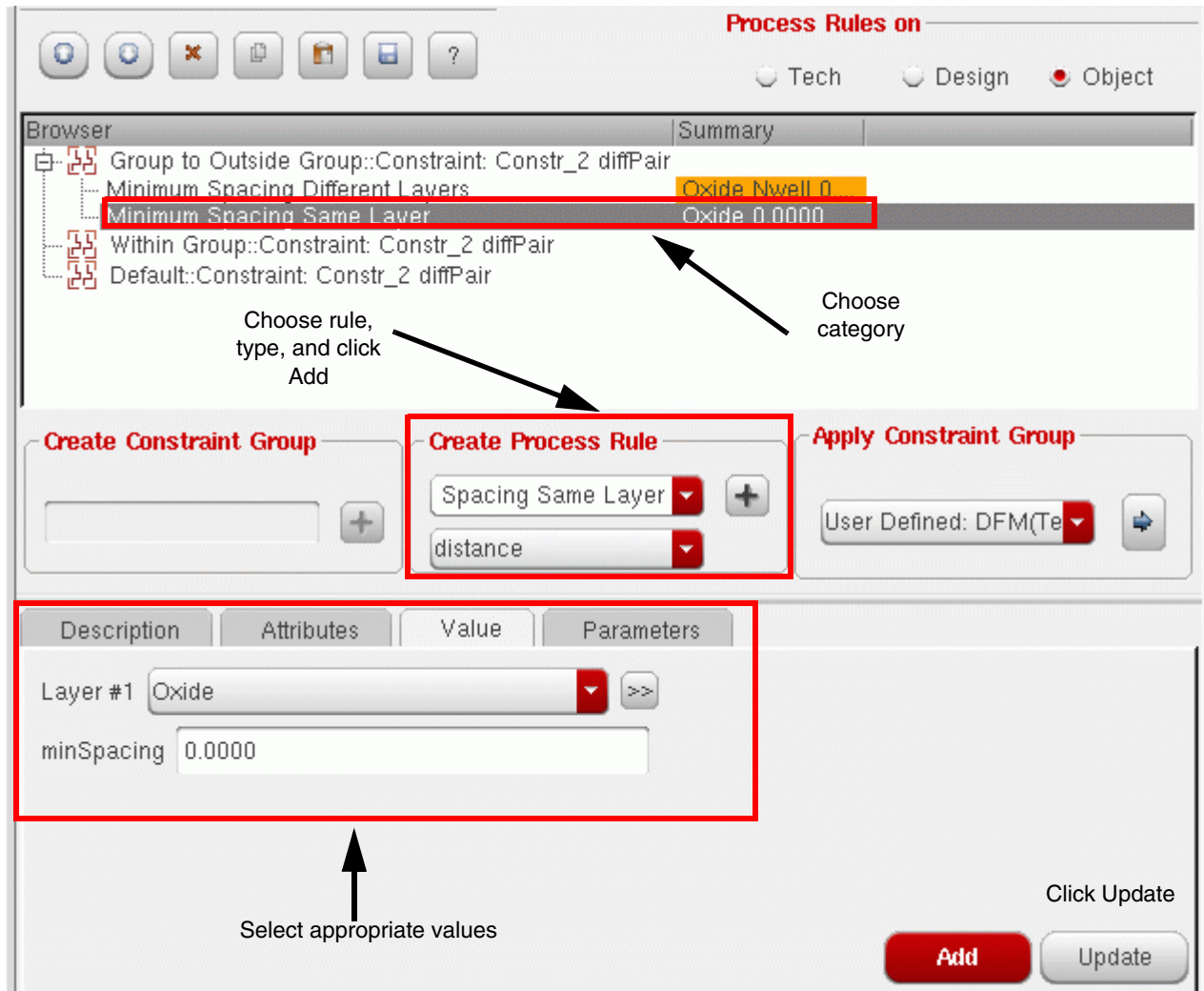
Note: Each list is seeded appropriately depending on what is selected in the browser pane.

5. Click the *plus (+)* button in the *Create Process Rule* section to add the new process rule to the selected constraint group.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Note: Alternatively, press the *Enter* key.



The process rule is added to the list. The *Summary* field in the browser pane is highlighted brown to indicate that the rule was created using the default values. You can modify the rule by changing the default values. This also removes the highlight from the *Summary* column.

6. Modify the rule by specifying appropriate data on the *Attributes*, *Value*, and *Parameters* tabs.

Note: Only relevant tabs are available for a process rule.

7. Click *Update* to apply the values you set in the *Value* tab.

Note: The changes you make on the *Attributes* and *Parameters* tabs are automatically saved. Therefore, you do not need to click the *Update* button to save these changes.

The new values that you specify on the *Value* tab appear in the *Summary* column of the browser pane.



Tip

To quickly add more process rules of the same type, click the *Add* button instead of *Update*. Each click adds a new rule with the same values displaying in the *Summary* column as the last one you added. The *Summary* column appears brown to indicate that the process rule values might need to be changed.

8. Select each new rule you added and edit the values, as required.

Copying an Existing Process Rule

You can quickly create new process rules by copying the existing ones.

Note: You can copy only Process Rules Supported by the Process Rule Editor. Any unsupported rules in the selected set are ignored.

To copy some existing process rules to a new location:

1. In the browser pane, select one or more process rules you want to copy.

Note:

- ☐ Hold the Shift key while you select multiple items in the tree.
- ☐ The selected set can contain multiple process rules from different constraint groups.

2. Click the *Copy Rules* button on the toolbar.

Note: Alternatively, right-click the process rule that you want to copy and choose *Copy Rules*.

The selected rules are copied to the clipboard.

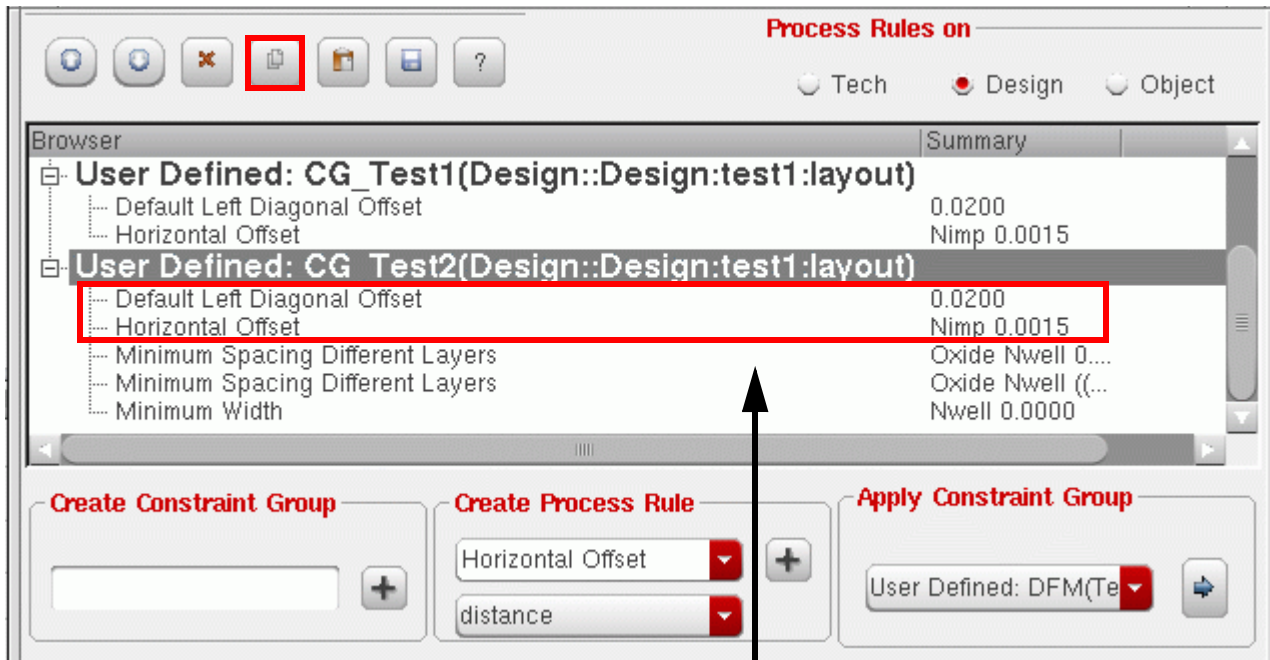
3. Select the constraint group to which you want to copy the selected rules and click the *Paste Rules* button on the toolbar.

Note: Alternatively, right-click the target constraint group and choose *Paste Rules*.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

The process rules are pasted into the new location as displayed in the figure below.



The rules are copied to
the new location

Note:

- ❑ You can paste the copied rules to a single constraint group at a time. The tool does not support selecting multiple target destinations for pasting the copied rules.
- ❑ To copy a rule to different target destinations, select each target at a time and click the *Paste Rules* button on the toolbar. Alternatively, you can use the *Paste Rules* command from the shortcut menu.

If you delete a process rule in the browser in between copying and pasting, the deleted rule is also removed from the clipboard and is not subsequently pasted.

Creating a Layer-Purpose Pair

To create a layer-purpose pair in PRE:

1. Select a layer from the drop-down list box and click the *arrow* (➤) button to the right of the list.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

A new purpose drop-down list box appears, as displayed in the figure below.

The screenshot shows the 'Process Rule Editor' interface. At the top, there are three main sections: 'Create Constraint Group', 'Create Process Rule', and 'Apply Constraint Group'. The 'Create Process Rule' section contains two parameters: 'Horizontal Routing Pitch' and 'distance'. Below these, there are tabs for 'Description', 'Attributes', 'Value', and 'Parameters'. The 'Value' tab is selected, showing 'Layer #1' set to 'Oxide' and 'Layer #2' set to 'Poly'. A 'minSpacing' field is set to '0.0000'. A red box highlights the 'Purpose' dropdown menu, which is currently set to 'NONE'.

Note: The default value for layer purpose is NONE. This indicates that a purpose is currently not defined for the layer.

2. Select an appropriate purpose, such as drawing, for the selected layer.

This screenshot shows the same 'Process Rule Editor' interface as the previous one, but with the 'Purpose' dropdown menu for 'Layer #1' set to 'drawing'. Above the editor, a 'Browser' window is open, displaying a list of constraints. The 'Minimum Spacing Different Layers' constraint is selected, and its 'Value' is '0.0000'. The 'Summary' tab is active, showing details for the selected constraint: 'Cont 0.1200', 'Metal1 0.2900', 'Oxide Poly 0.0000', 'Metal1 0.2900', 'Metal1 0.1450', '0.0000', 'Metal1 0.1450', and 'Metal1 Metal2 Metal3 Met...'. The 'Process Rule Editor' at the bottom shows the 'Value' tab with 'Layer #1' set to 'Oxide' and 'Layer #2' set to 'Poly'. The 'minSpacing' field is '0.0000'. A red box highlights the 'Purpose' dropdown menu, which is now set to 'drawing'.

Note: Currently, PRE supports default purposes for most of the layer constraints. However, other Virtuoso applications fully support purposes only for the following constraints: *Minimum Spacing*, *Minimum Width*, and *Minimum Clearance*. Therefore, the default state of the *Purpose* cyclic field for the constraints that are not

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

“purpose-supported” by other Virtuoso applications is `NONE`.

For more information about the layer-purpose pairs supported by the Virtuoso technology file, see [Purpose-Aware Derived Layers](#) and [techPurposes](#) in the *Virtuoso Technology Data ASCII Files Reference User Guide*.

3. Specify an appropriate data format value for the selected layer. For example, when creating a *Minimum Spacing Different Layers* rule, you need to specify a minimum spacing value, such as `0.0250`, between the two selected layers.

The screenshot displays the Virtuoso Process Rule Editor interface. At the top, a 'Browser' pane lists various constraint types, with 'Minimum Spacing Different Layers' selected and highlighted in red. The 'Summary' pane shows the configuration for this rule: 'Oxide-drawing Poly-fill 0.0250'. Below the browser, there are three main sections: 'Create Constraint Group', 'Create Process Rule', and 'Apply Constraint Group'. The 'Create Process Rule' section shows the rule name 'Spacing Different Layers' and the number of layers '2 Layers, distance'. The 'Apply Constraint Group' section shows the user-defined group 'DFM(Te)'. Below these sections, there are tabs for 'Description', 'Attributes', 'Value', and 'Parameters'. The 'Value' tab is active, showing the 'minSpacing' attribute set to '0.0250'. At the bottom right, there are 'Add' and 'Update' buttons, with the 'Update' button highlighted in red.

Browser	Summary
Horizontal Pitch	Metal1 0.2900
Vertical Pitch	Metal1 0.2900
Minimum Spacing Different Layers	Oxide-drawing Poly-fill 0.0250
Horizontal Offset	Metal1 0.1450
Default Horizontal Routing Pitch	0.0000
Vertical Offset	Metal1 0.1450
Valid Layers	Metal1 Metal2 Metal3 Metal...
Valid Vias	M1_POV M2_M1v M3_M2v...
Minimum Spacing Different Layers	Oxide Nwell 0.0000
Minimum Spacing Different Layers	Oxide Nwell 0.0000

Create Constraint Group: [] +

Create Process Rule: [Spacing Different Layers] + [2 Layers, distance]

Apply Constraint Group: [User Defined: DFM(Te)] +

Layer #1: [Oxide] >> Layer #2: [Poly] >>

minSpacing: [0.0250]

Add Update

4. Click *Update* to apply the new values you set on the *Value* tab.

The *Summary* column displays a single concatenated string representing the layers used for the selected process rule and the current value of the constraint. The brown color indicates that the process rule has been created using default values, which might need to be changed.

Applying Valid LPPs Process Rule

In addition to allowing purposes to be defined for the selected layer when creating a process rule, the Process Rule Editor supports the *Valid LPPs* (Valid Layer Purpose Pairs) process rule. This rule defines the valid layer-purpose pairs that can be used for routing. Like most other process rules, you can add this process rule to a constraint group in the *Technology* or *Design* scopes or to an *Object*. You can also modify or delete the process rule, as required.

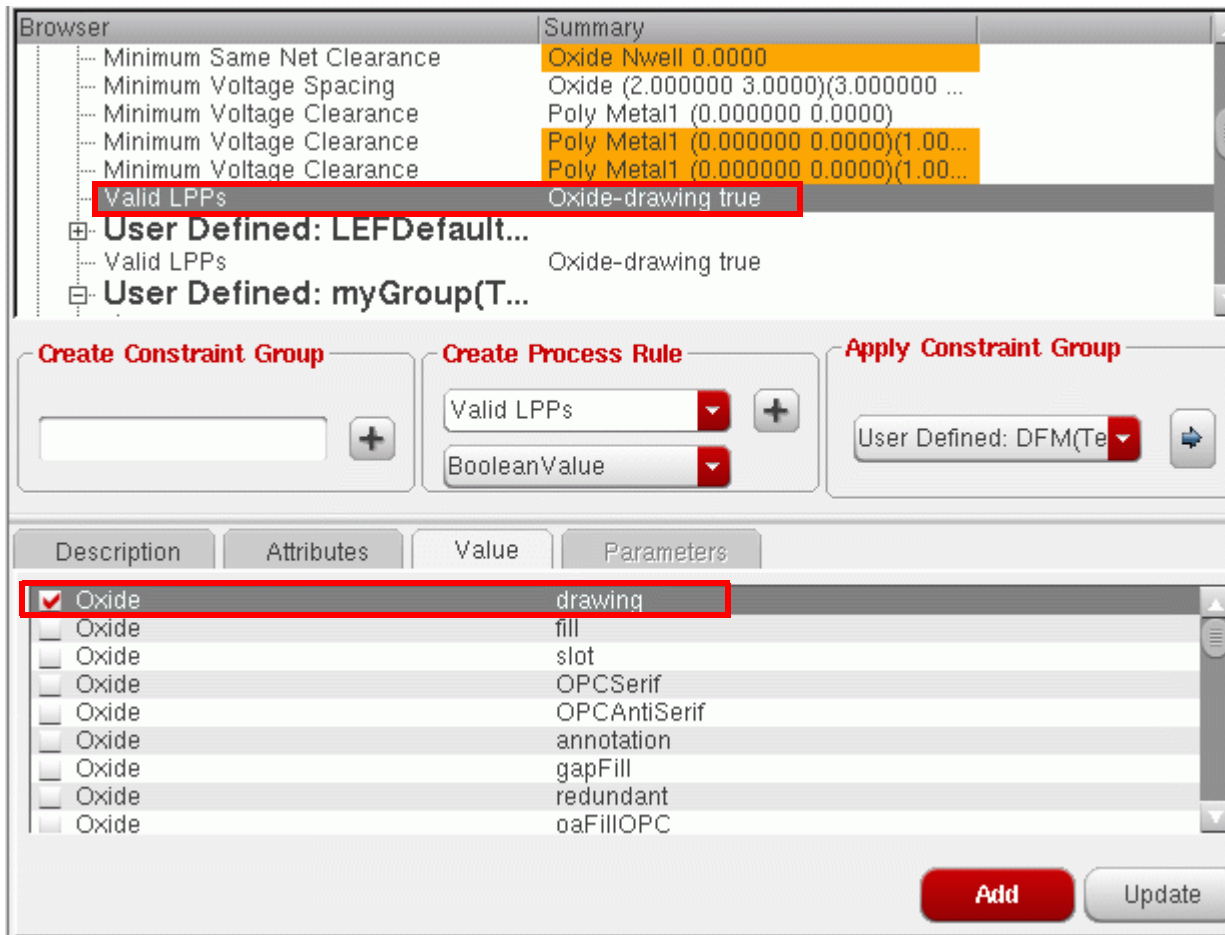
To select a valid layer-purpose pair from the Valid LPPs process rule:

1. Select the *Valid LPPs* process rule in the browser pane and click the *Value* tab.
2. From the list of layer-purpose pairs displayed on the *Value* tab, select the check box next to the layer name in the row that carries the required layer-purpose combination.
3. Click *Update* to apply the layer-purpose values you selected.

The layer-purpose pair that you selected displays as a single, hyphenated string in the *Summary* column of the browser pane.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor



For information about creating a layer-purpose pair in PRE, see [Creating a Layer-Purpose Pair](#).

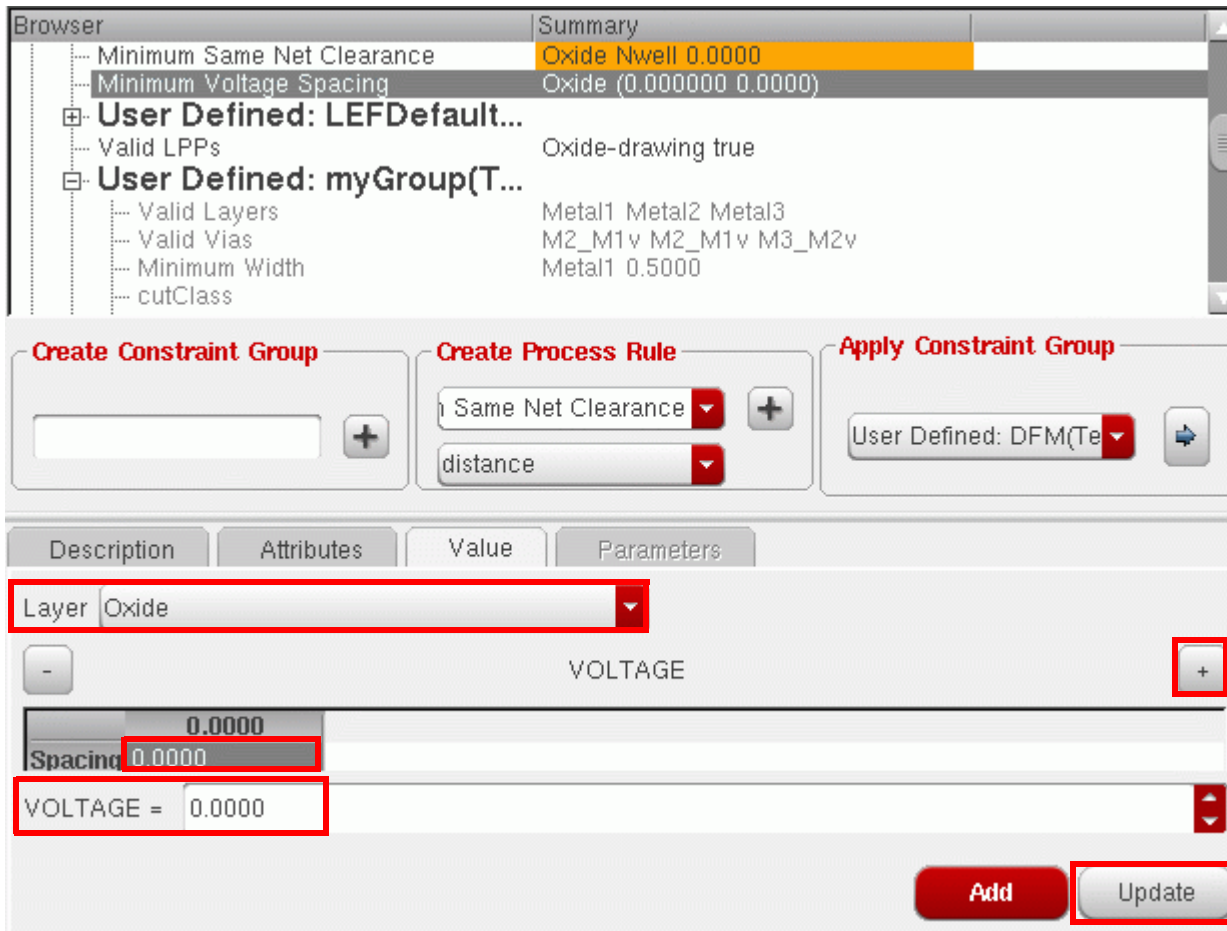
Creating a Voltage Dependent Process Rule

Voltage dependent rules are the ones that define a process constraint in terms of a voltage level. Currently, PRE supports two such rules—`minVoltageSpacing` and `minVoltageClearance`. Both these rules define the separation constraint between two devices in terms of voltage levels. The data value format supported by both the rules is voltage, spacing.

`minVoltageSpacing` defines the spacing between devices along a single layer, whereas `minVoltageClearance` defines the spacing between devices on two layers. You can define the required voltage values and the corresponding spacing for both the process rules using the VOLTAGE-Spacing table in the *Value* tab, as displayed below.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor



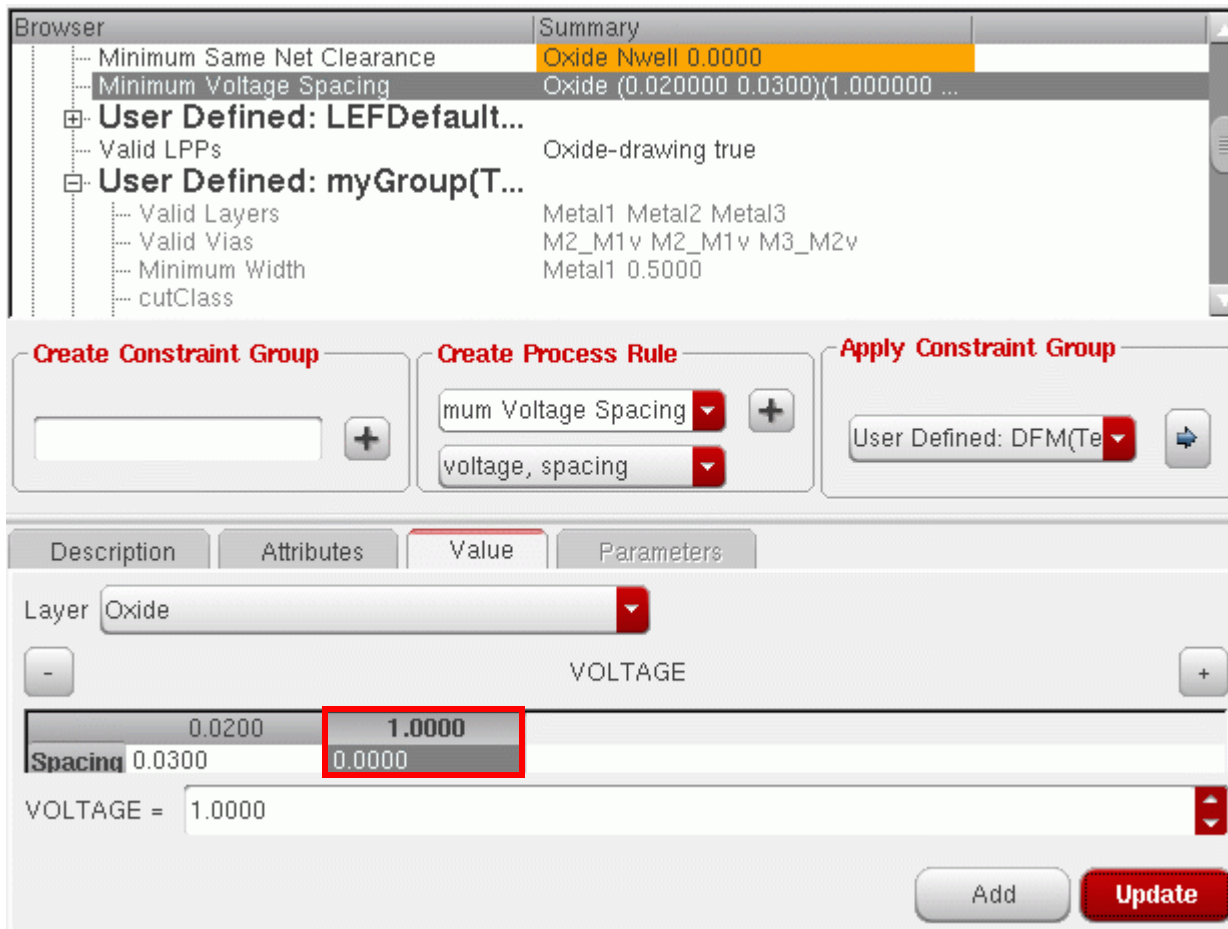
To specify the required voltage and spacing values for a selected layer:

1. Type an appropriate value for the voltage in the *VOLTAGE* field.
2. Double-click the *Spacing* field and type in an appropriate value.
3. To add another voltage-spacing column to the table, click the *Insert new column (+)* button on the right.

A new VOLTAGE-Spacing column is added as displayed below.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor



If you click the Insert new column (+) button multiple times, multiple columns are added. By default, the voltage in each column increments by 1. You can select each column and modify the voltage and spacing values, as required.

4. Click *Update* to apply the values you set.

The new values that you specify on the *Value* tab appear in the *Summary* column of the browser pane.

For `minVoltageClearance`, you need to select two layers instead of one and perform the steps as described above.

Editing Existing Process Rule Values

To edit the layer-specific values for an existing process rule:

1. In the browser pane, select the process rule you want to change.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

A dynamic widget is displayed. The widget comprises the following tabs:

- ☐ Description Tab
- ☐ Attributes Tab
- ☐ Value Tab
- ☐ Parameters Tab

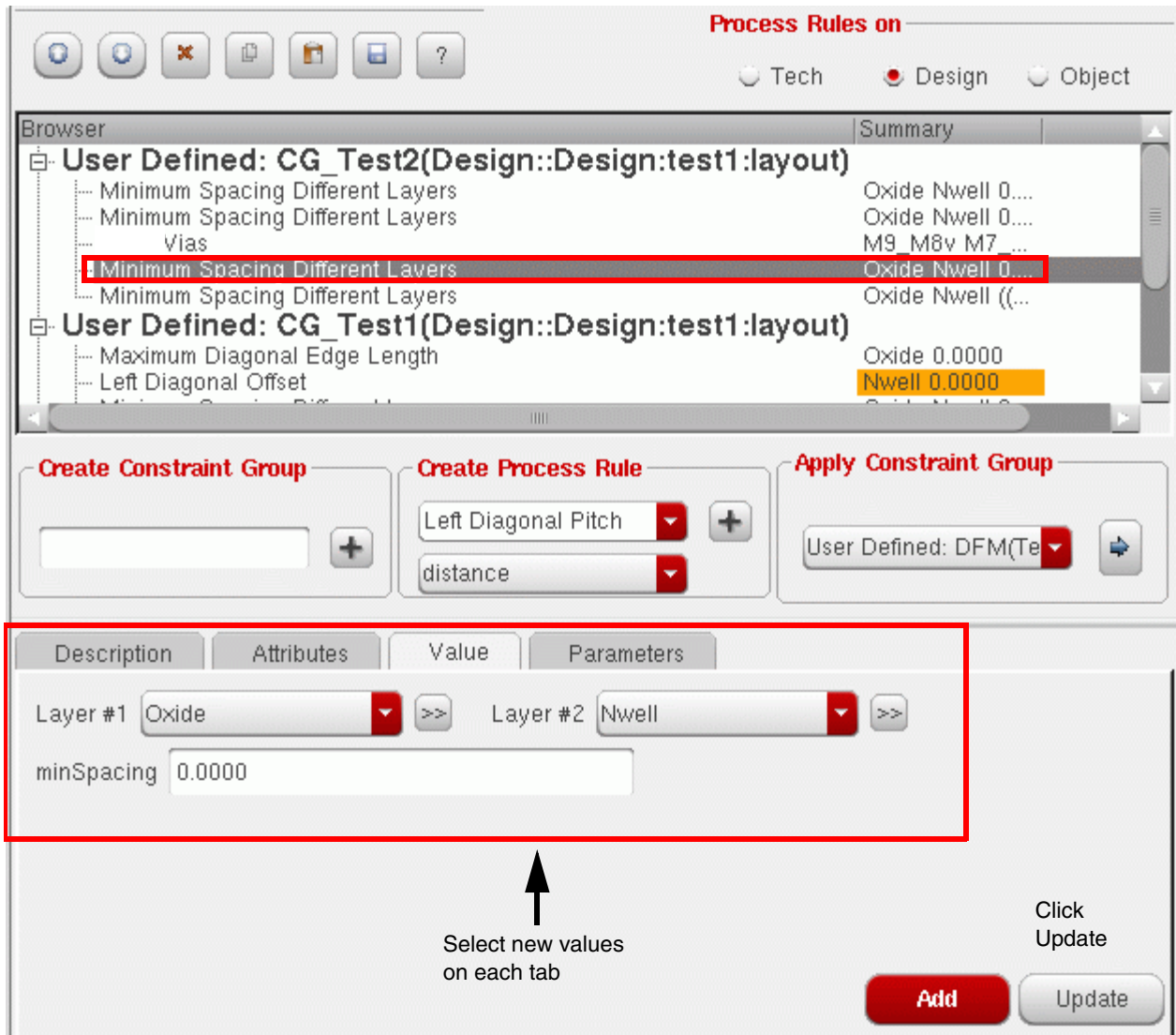
Each tab displays the current values for the selected process rule.

2. To edit the existing process rule values, change the values in each of the tabs, as required.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Note: The *Description* tab is not editable.



3. Click the *Update* button to save the values you changed on the *Value* tab.

Note: The changes you make on the *Attributes* and *Parameters* tabs are automatically saved. Therefore, you do not need to click the *Update* button to save these changes.

The process rule is updated to reflect the new settings.



To quickly add more process rules of the same type, click the *Add* button instead of *Update*. Each click adds a new rule with the same values displaying in the *Summary* column as the last one you added. The *Summary* column appears brown to indicate that the process rule values might need to be changed.

Applying a Constraint Group

For a process rule to be applied to a technology, design, or object; it must be a member of a constraint group that supports the rule. Likewise, when applying a constraint group, the target constraint should support the applied constraint group.

For example, when applying a Constraint Group in the Technology scope, you can only select the Constraint Groups that are supported by the Technology database. The supported items are the ones that appear in the Constraint Group drop-down menu for each database.

Note: Constraint Groups cannot be made nested. For example, if you apply *Constraint Group A* to *Constraint Group B*; you cannot then apply *Constraint Group B* to *Constraint Group A*. Such nested applications are not supported by PRE.

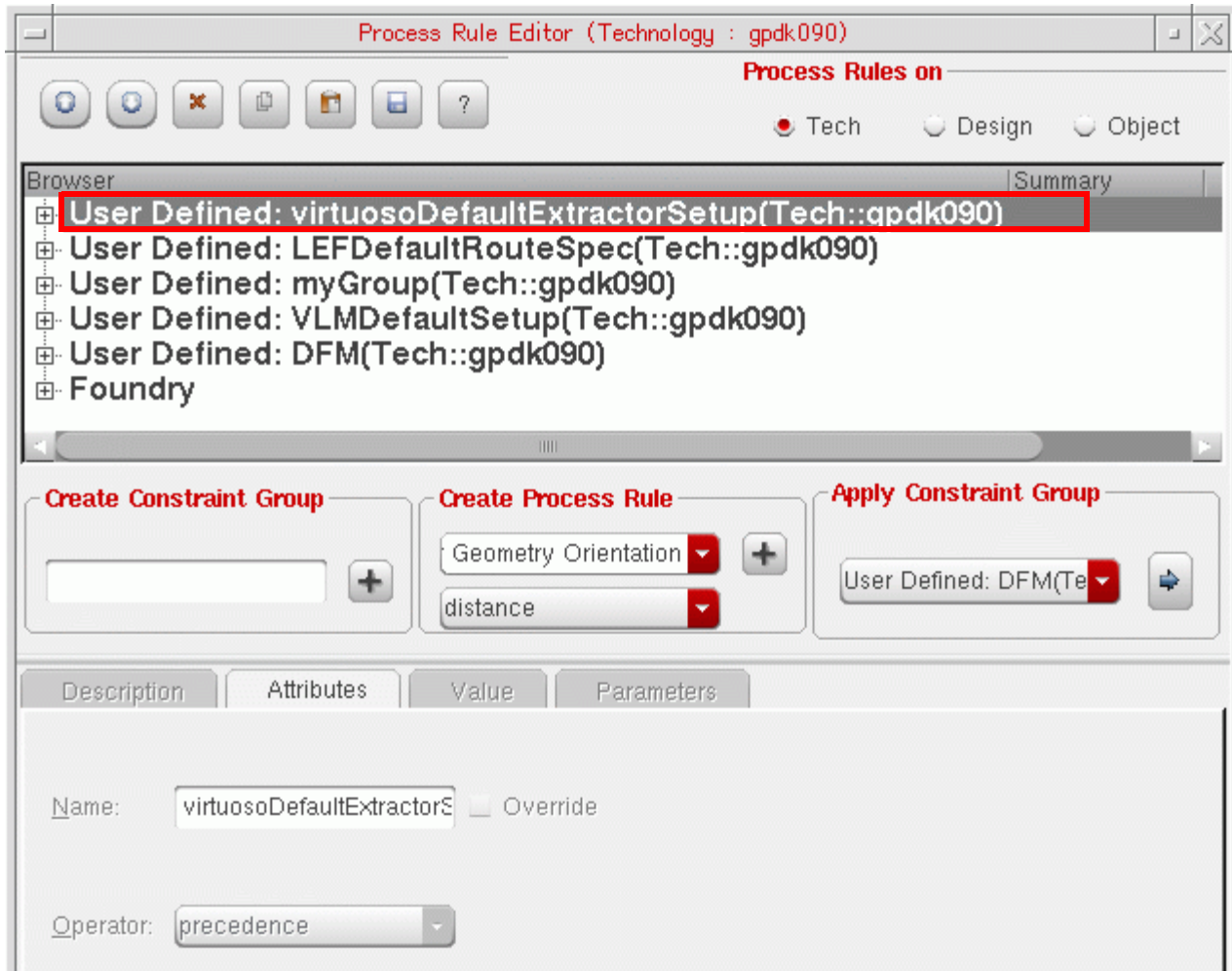
To apply a constraint group to another constraint group, use the data value formats in the **Apply Constraint Group** section. The *Constraint Group* drop-down menu reflects the associated database as a tag (Tech or Design). This helps avoid confusion when selecting the constraint groups.

Let us now apply a constraint group to the *virtuosoDefaultExtractorSetup* constraint group in the browser.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

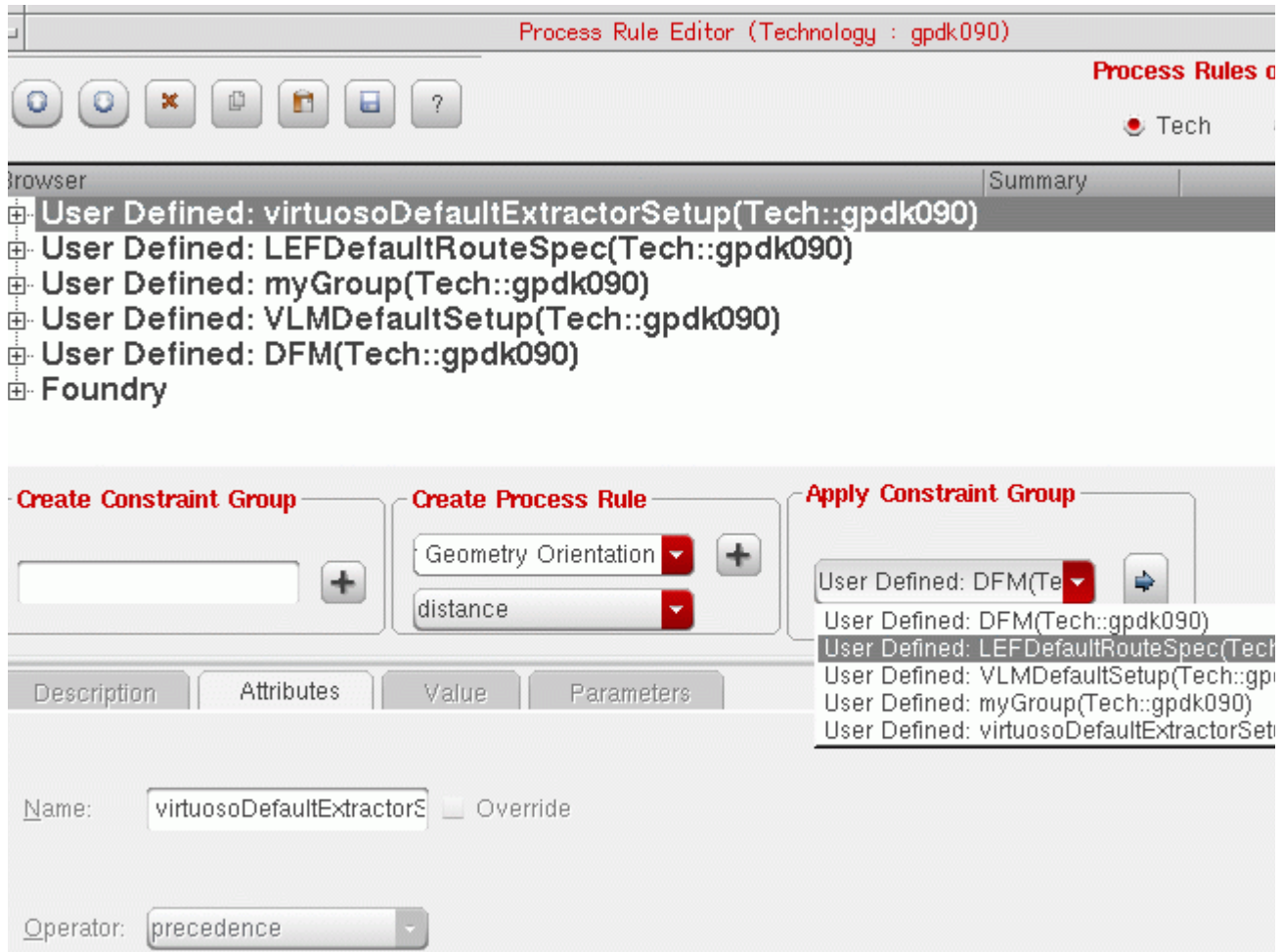
1. Select the *virtuosoDefaultExtractorSetup* constraint group in the browser pane.



Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

2. Choose the constraint group you want to apply from the drop-down list box in the **Apply Constraint Group** section.

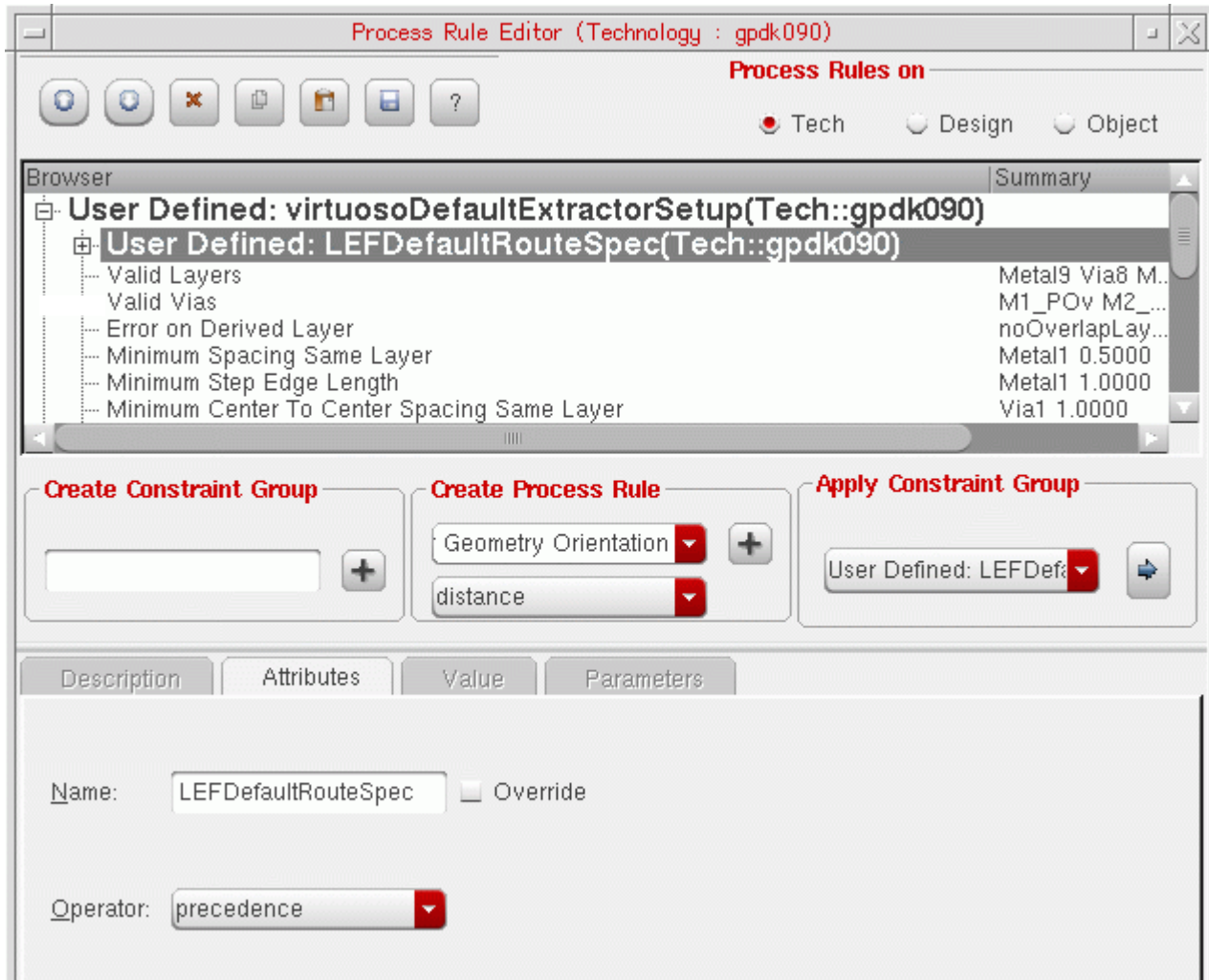


3. Click the arrow button on the right of the list box to apply the constraint group.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

The constraint group is inserted into the *virtuosoDefaultExtractorSetup* constraint group in the browser pane. Any process rules defined in the *DefaultRouteSpec* constraint will now be applied to any design referencing the *virtuosoDefaultExtractorSetup* constraint group.



The constraint group is still also listed on the main branch of the browser pane.

To make any changes to the Constraint Group, you must update the source Constraint Group and not the one linked in. Applied constraint groups are not available for editing. In fact, any changes you make to the source automatically reflect at instances where the constraint group is linked.

For example, to update the process rules for *LEFDefaultRouteSpec*, you must update the constraint group in the main browser pane instead of trying to update the instance appearing under *virtuosoDefaultExtractorSetup*.

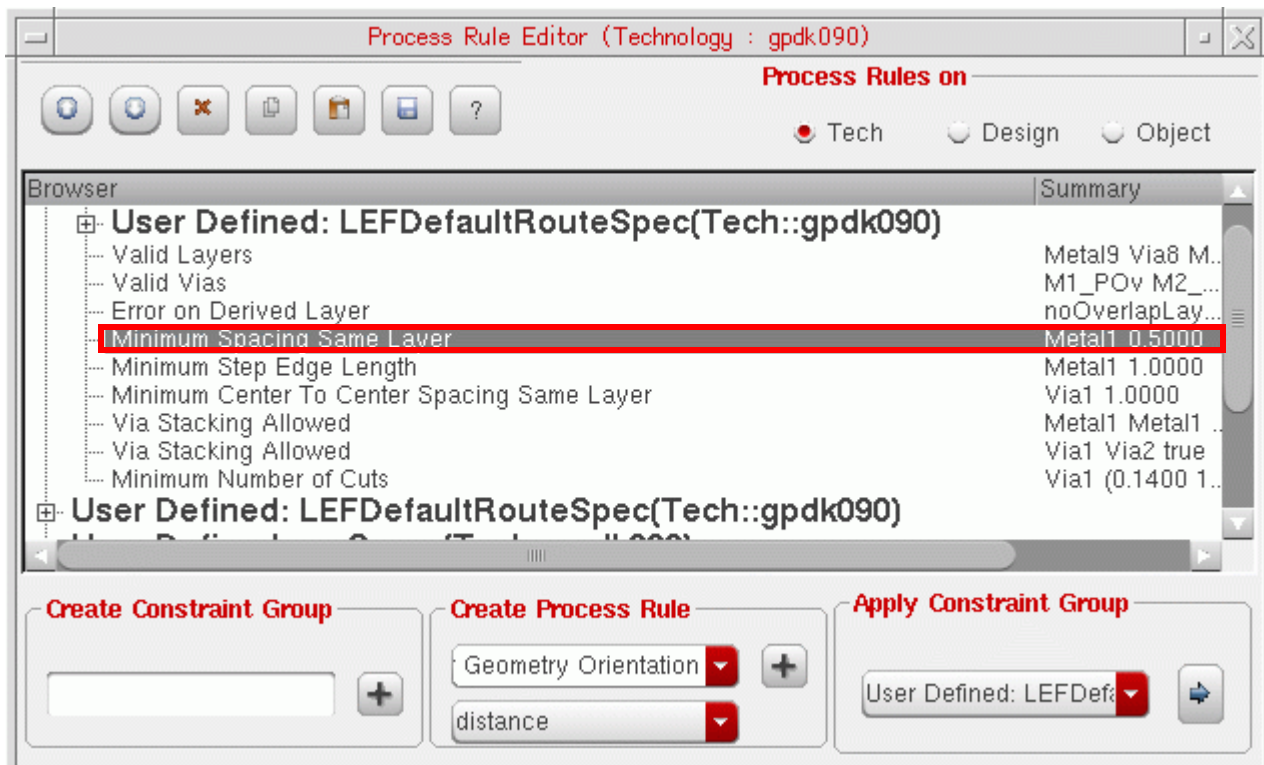
Setting the Precedence of a Constraint Group or Process Rule

Process rules are applied in the order in which they are listed inside a constraint group. However, you can change the precedence of an applied constraint group or individual process rule by using the *Move Up* and *Move Down* arrow buttons on the toolbar.

Note: The order of the constraint group at the top level of the browser tree is not significant, so the arrow buttons are inactivated when a top-level constraint is selected.

To change the precedence of an applied constraint group or process rule:

1. Select the constraint group or process rule you want to move.



2. Do one of the following:

- ☐ Hold down the left mouse button to drag the constraint group to its new location. This method lets you move the selected constraint group or process rule more than one position at a time.
- ☐ Right-click the constraint group or process rule and choose *Move Up* to increase or *Move Down* to decrease the precedence. This moves the selected object up or down the list one position at a time.

Virtuoso Unified Custom Constraints User Guide

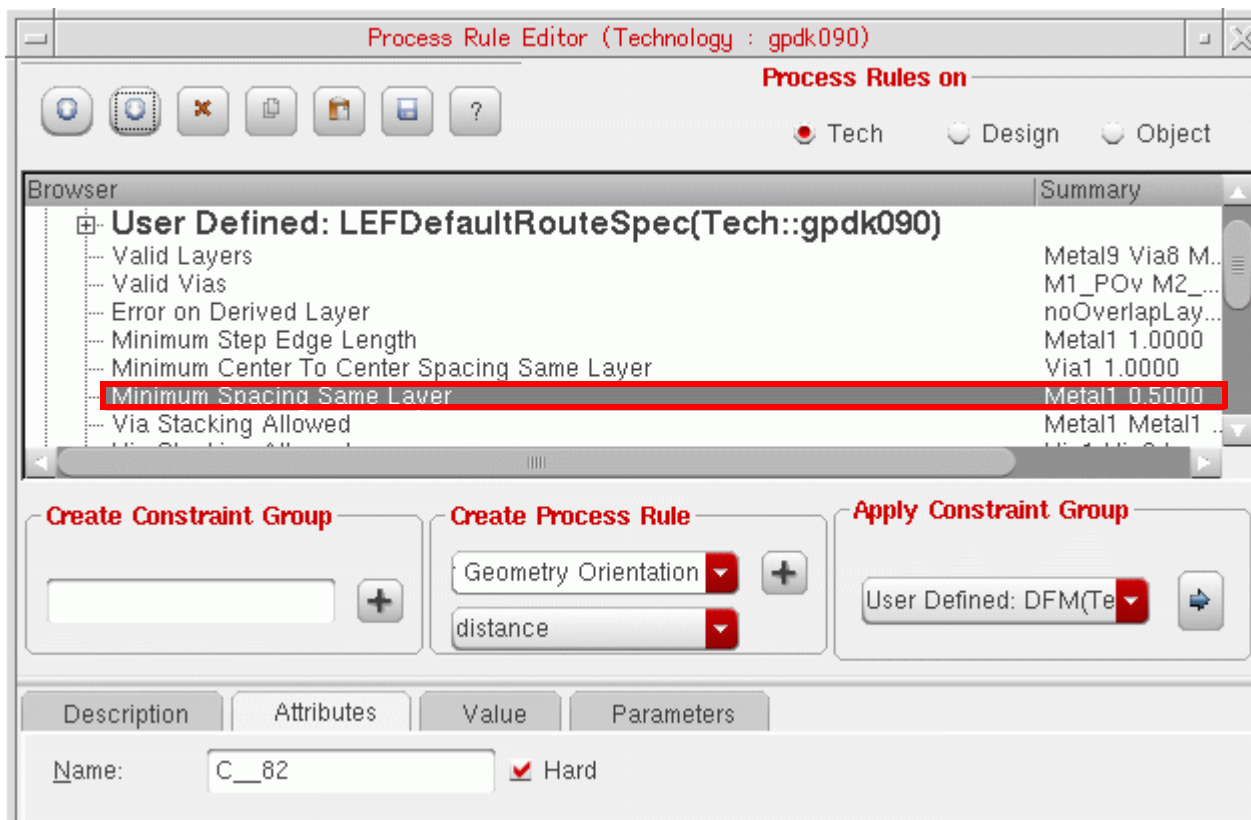
The Process Rule Editor

Note: These functions are not available for objects at the top level of the browser tree comprising the Constraint Groups.

- ❑ Click the *Move Up* or *Move Down* button on the Process Rule Editor toolbar. This moves the selected object up or down the list one position at a time.



The browser pane updates to show the constraint group or process rule at its new location.



Virtuoso Generated Constraint Groups

Virtuoso ships with a number of predefined constraint groups, which you can generate and apply to selected objects or constraints in your design.

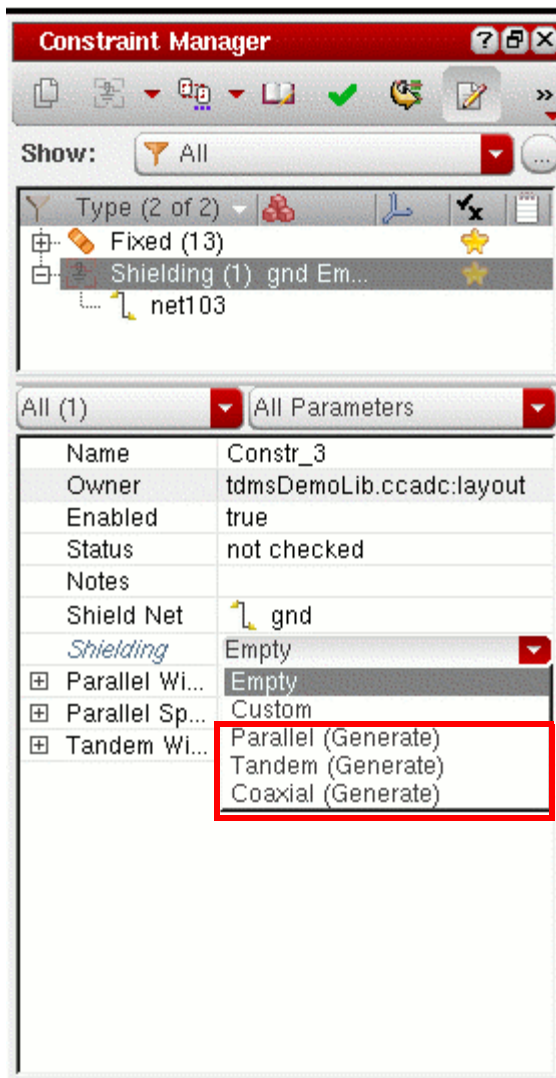
These constraint groups are generated with a predefined set of rules set to values that are based on the available technology information. A generated constraint group is stored in the

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

design when you apply it to a selected object or constraint in that design. The constraint group can also be applied to other objects and constraints. This lets you quickly generate and reuse constraints groups without having to manually enter the long list of common process rules. Furthermore, this allows the users to create their own Constraint Groups and modify them, as required, in less time. You can generate a predefined constraint group from the *Constraint Manager* parameters pane. To do this, select the object or constraint to be constrained, generate the type of constraint group you need, and then add, modify, or remove the process rules, as required.

For example, the picture below shows the constraint groups you can generate for a *Shielding* constraint – *Parallel*, *Tandem*, and *Coaxial*.



The label *Generate* at the end of each constraint group name indicates that this is a predefined constraint group that has not yet been generated for the current design. When you

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

choose a constraint group from the *Shielding* drop-down list box, that group is generated with its default rules in the Process Rule Editor browser pane. This causes the *Generate* entry to be removed from the list box and is replaced by a *Design* entry of the same name. This indicates that the constraint group has been generated and stored in the design.

The table below lists the predefined constraint groups you can generate for each constraint type.

Constraint Type	Parameter Name	Constraint Group	Contents
<i>Diff Pair</i>	<i>Within Group</i>	<i>2x Spacing thru 10x Spacing</i>	Varying multiples of the minimum spacing defined for each layer in the reference technology file.
	<i>Group to Outside Group</i>	<i>2x Spacing thru 10x Spacing</i>	Varying multiples of the minimum spacing defined for each layer in the reference technology file.
<i>Matched Length</i>	<i>Group to Outside Group</i>	<i>2x Spacing thru 10x Spacing</i>	Varying multiples of the minimum spacing defined for each layer in the reference technology file.
<i>Bus Net</i>	<i>Within Group</i>	<i>2x Spacing thru 10x Spacing</i>	Varying multiples of the minimum spacing defined for each layer in the reference technology file.
	<i>Group to Outside Group</i>	<i>2x Spacing thru 10x Spacing</i>	Varying multiples of the minimum spacing defined for each layer in the reference technology file.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Constraint Type	Parameter Name	Constraint Group	Contents
<i>Process Rule Override</i>	<i>Default</i>	<i>2x Width thru 10x Width</i>	Varying multiples of the minimum width specified for each layer in the reference technology file.
	<i>Input Taper</i>	<i>1UU Taper Halo</i>	Taper halo set to 1u. Allowed layers are the same as those defined in the reference technology file plus all polysilicon materials.
	<i>Output Taper</i>	<i>1UU Taper Halo</i>	Taper halo set to 1u. Allowed layers are the same as those defined in the reference technology file plus all polysilicon materials.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Constraint Type	Parameter Name	Constraint Group	Contents
<i>Shielding</i>	<i>Shielding</i>	<i>Parallel</i>	minWidth (shieldWidth) and minSpacing (shieldSpacing) for all the layers defined in the reference technology file. Width and spacings taken from the reference technology file.
		<i>Tandem</i>	<i>Tandem Layer Above</i> specifies the layer above the given signal wire layer to use as a shielding (Layer#1= signal Layer#2 = Shielding) <i>Tandem Width</i> specifies the width for the layer in the shielding constraint.
		<i>Coaxial</i>	Parallel rules created as above, tandem rules created as above but wider to cover both the signal and parallel shields.
<i>Symmetry</i>	<i>Group to Outside Group</i>	<i>2x Spacing thru 10x Spacing</i>	Varying multiples of the minimum spacing defined for each layer in the reference technology file.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

Constraint Type	Parameter Name	Constraint Group	Contents
<i>Net Class</i>	<i>Within Group</i>	<i>2x Spacing thru 10x Spacing</i>	Varying multiples of the minimum spacing defined for each layer in the reference technology file.
	<i>Group to Outside Group</i>	<i>2x Spacing thru 10x Spacing</i>	Varying multiples of the minimum spacing defined for each layer in the reference technology file.

Applying a Virtuoso Generated Constraint Group

You have a design which you want to route with a minimum width that is double the width defined in the technology files. You need to create and apply a constraint group that contains process rules re-specifying the minimum width for all the valid routing layers in your design.

You can either create this constraint group manually and add and set the value for each rule individually, or you can have Virtuoso generate a constraint group containing the rules you need. To do this,

1. On the Constraint Manager toolbar, click the *Process Rule Editor* button.



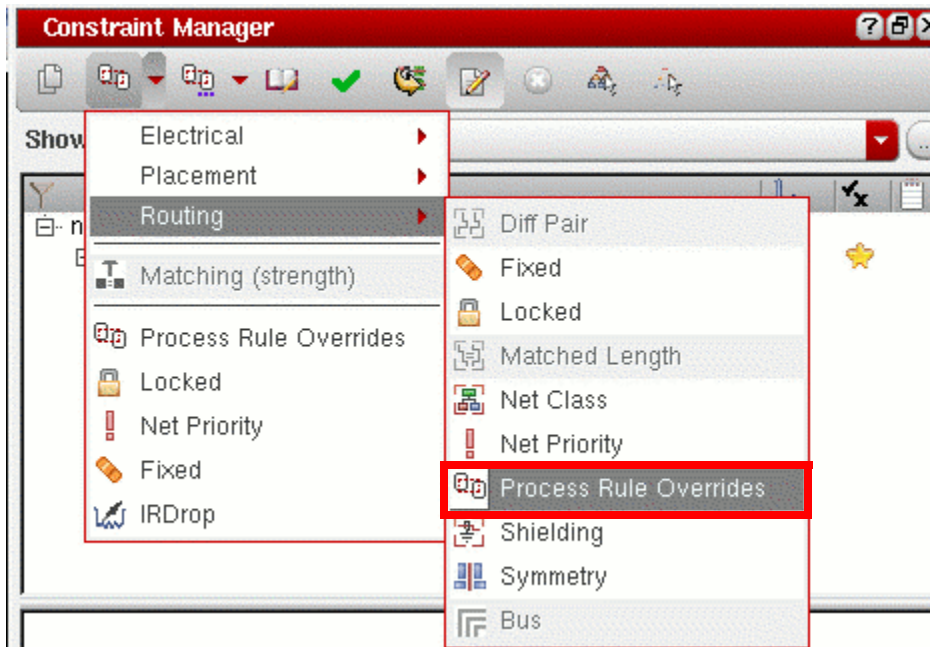
The Process Rule Editor Graphical User Interface is displayed.

2. From the Navigator assistant, select an object in the design.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

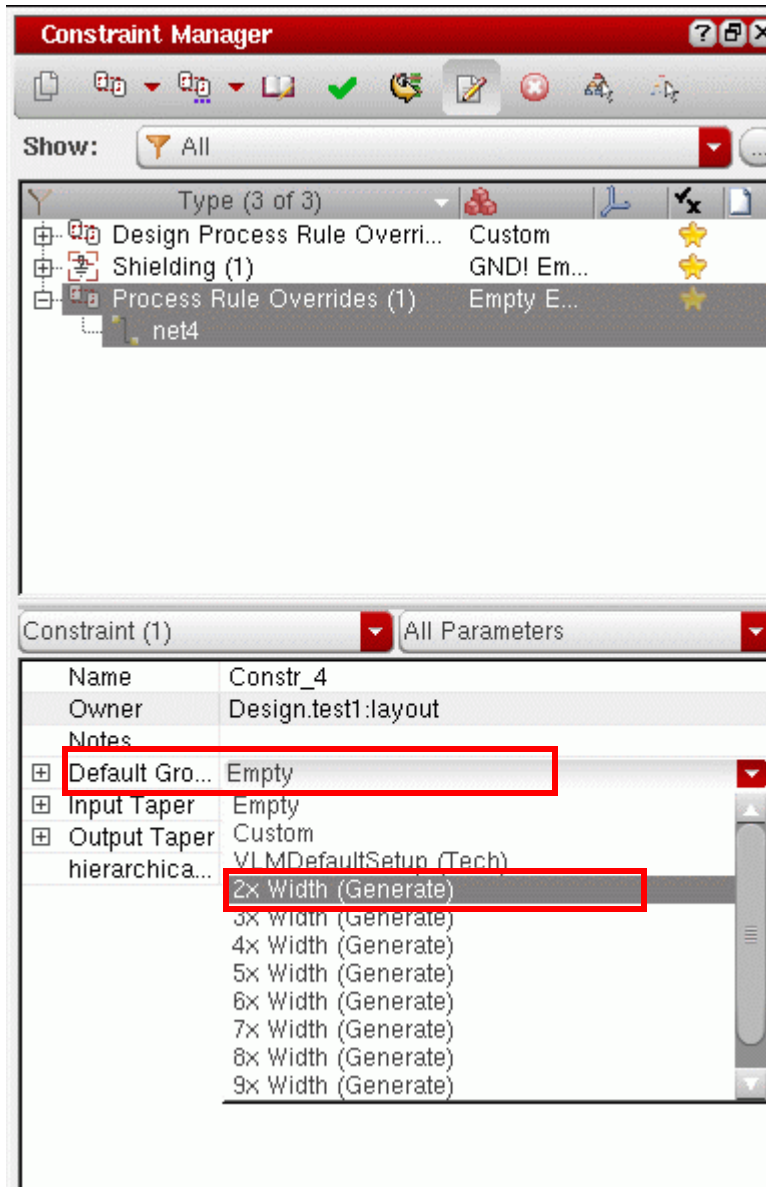
3. In the Constraint Manager assistant, create a *Process Rule Override* constraint for that object.



Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

4. Pull down the list next to the *Default* parameter and choose the *2x Width* generated constraint group.

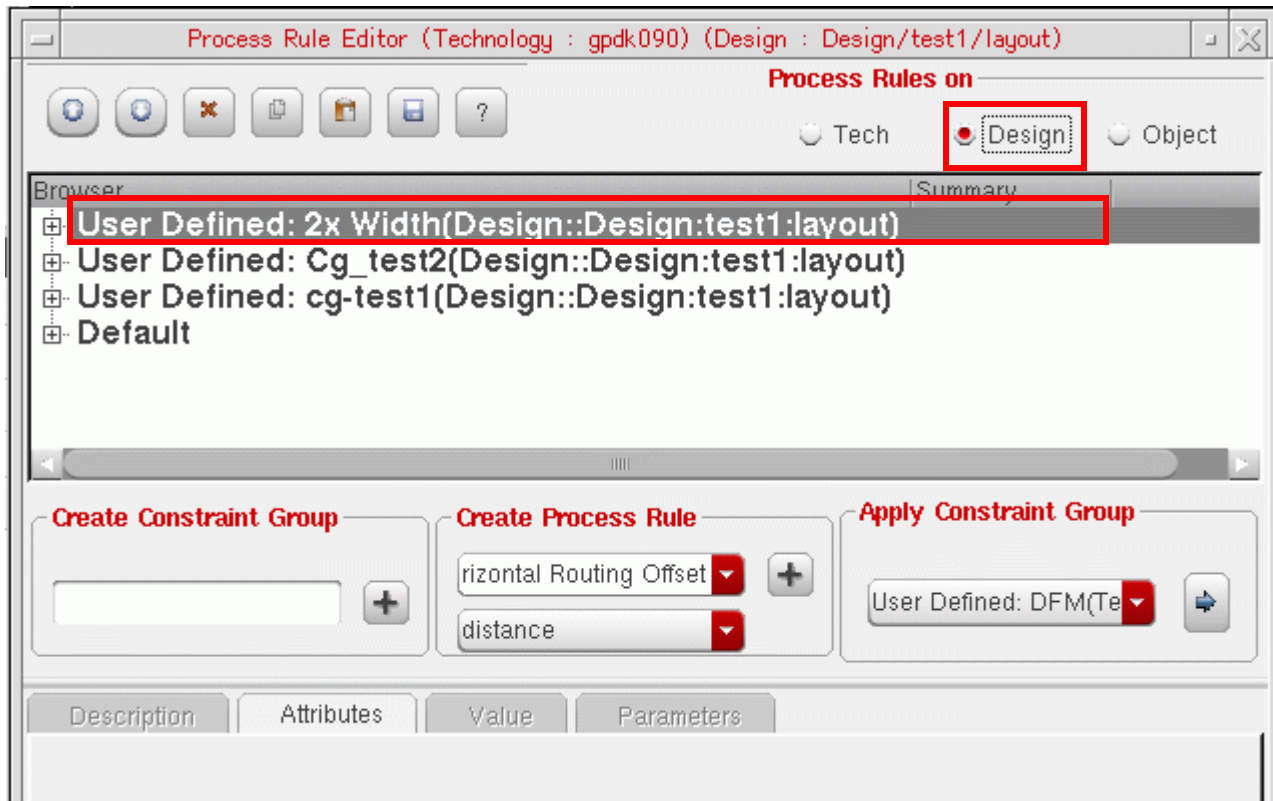


The *2x Width* constraint group contains rules that redefine the minimum width for each routing layer to be double the value defined in the technology files.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

The predefined constraint group is added to the *Design* scope of the Process Rule Editor.

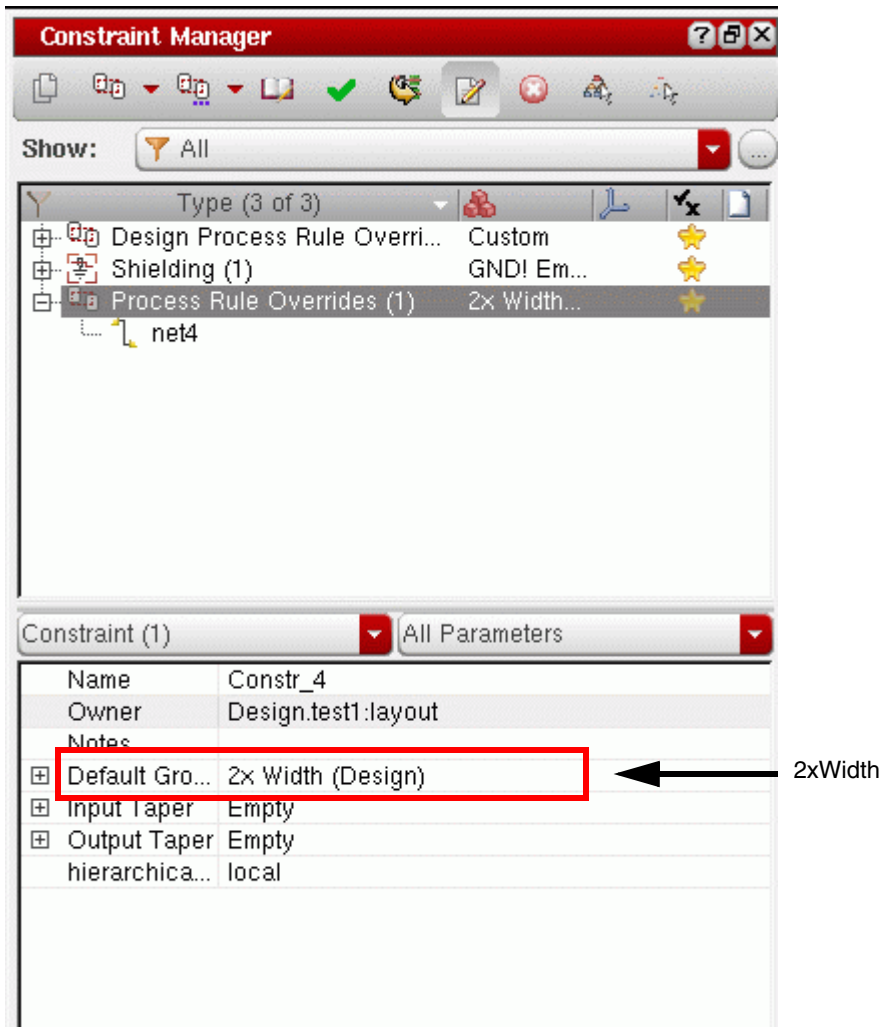


You can edit the constraint group directly or use the copy and paste functions to make the predefined constraint group more specific for your design.

For more information, see

- ❑ [Creating a New Process Rule in the Technology or Design Mode](#)
- ❑ [Copying an Existing Process Rule](#)
- ❑ [Editing Existing Process Rule Values](#)

If you now check the *Default* parameter drop-down list box in the *Constraint Manager*, you will see that the *Generate* entry has changed to the corresponding *Design* entry. This indicates the constraint group has been generated and added to the Design library. You can add this constraint group to any net in the Design or Object scope other than *net102* in the example.



Applying an Existing Constraint Group to a Selected Object

In addition to generating a new constraint group for an object, you can reapply an existing constraint group to another object. The parameters pane in the Constraint Manager assistant allows you this feature.

Assume that you have nets in your design. You want to route these nets using double-cut vias and with a minimum width larger than that defined in the technology files. To do this,

1. On the Constraint Manager toolbar, click the *Process Rule Editor* button ().

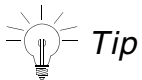
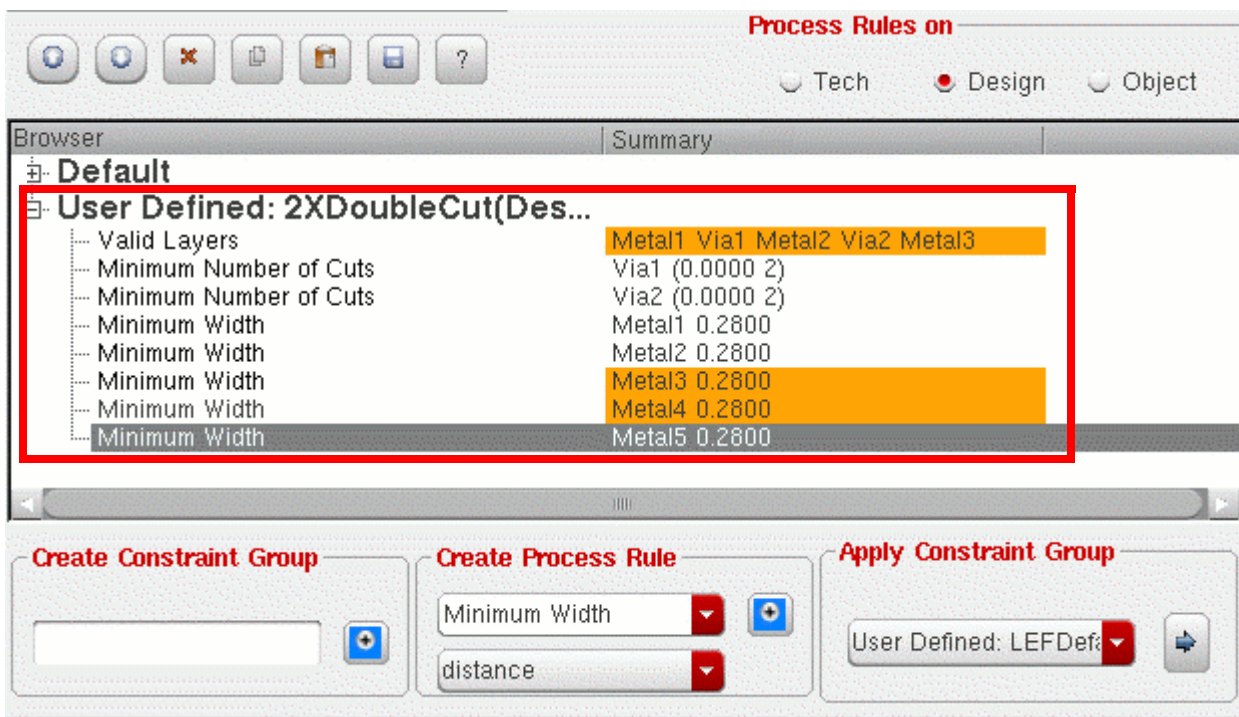
Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

The Process Rule Editor Graphical User Interface is displayed.

2. Create a constraint group containing the individual constraints you need.
 - a. Choose *Design* in the **Process Rules on** section.
 - b. Create a constraint group to contain the process rules you require. See Creating a Constraint Group.
 - c. Add the process rules you require. See Creating a New Process Rule.

You have created a constraint group definition such as the *2xDoubleCut* constraint group shown below, which specifies the list of valid layers, double cuts for each via and a larger width than the minimum specified in the technology file.



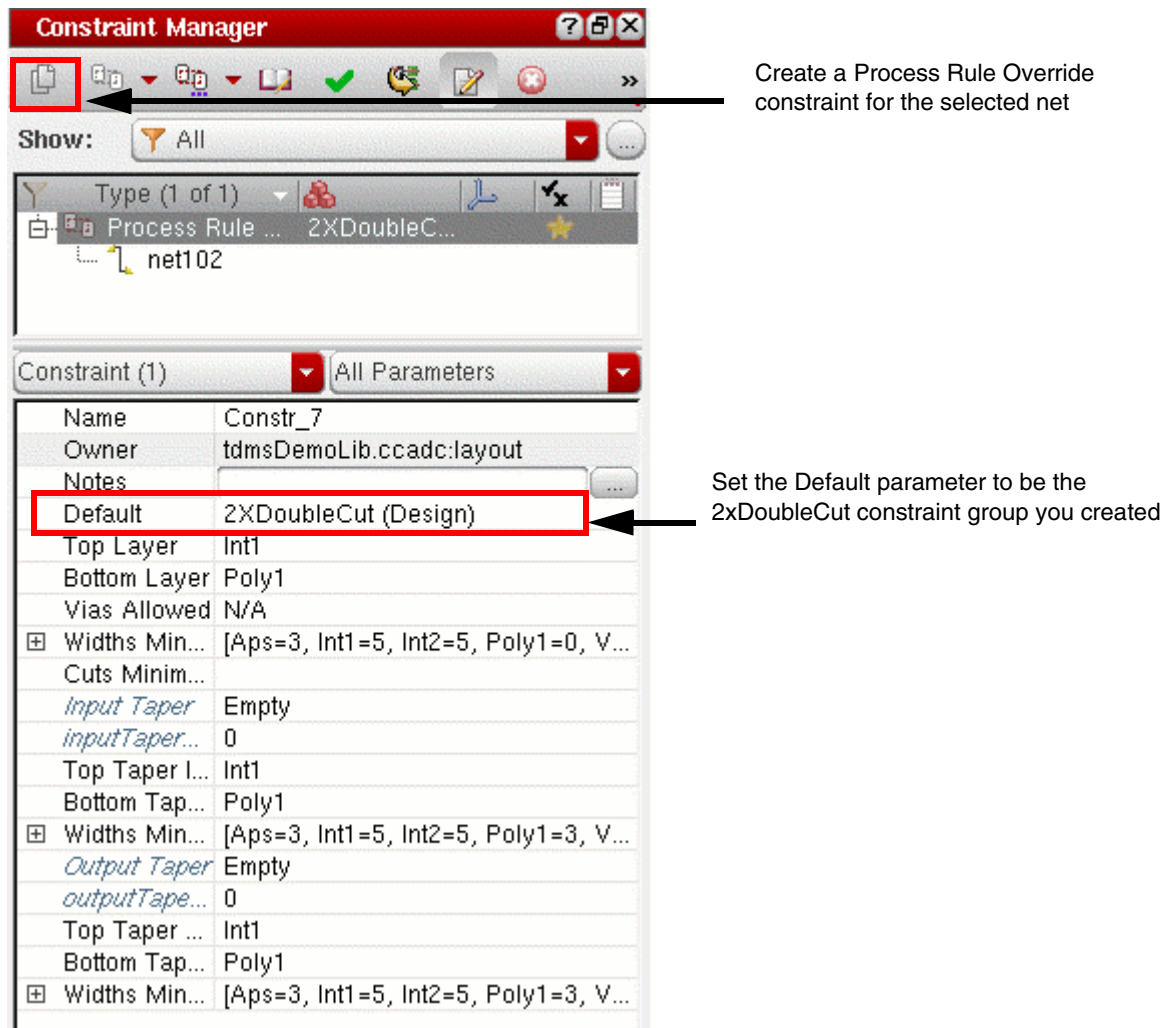
Tip

You could generate a constraint group like the one above more quickly by using one of the Virtuoso generated constraint groups as a starting point and adding the specific rules you need. For example, you could start with the *2xWidth* constraint group and then edit the generated group to limit the layers and create double cuts for all the vias. For more information, see Applying a Virtuoso Generated Constraint Group.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

3. From the Navigator assistant, select the first net to which you want to apply this constraint group.
4. In the Constraint Manager assistant, create a *Process Rule Override* constraint for the selected net.
5. Pull down the list next to the *Default* parameter and choose the *2xDoubleCut* constraint group you created previously.

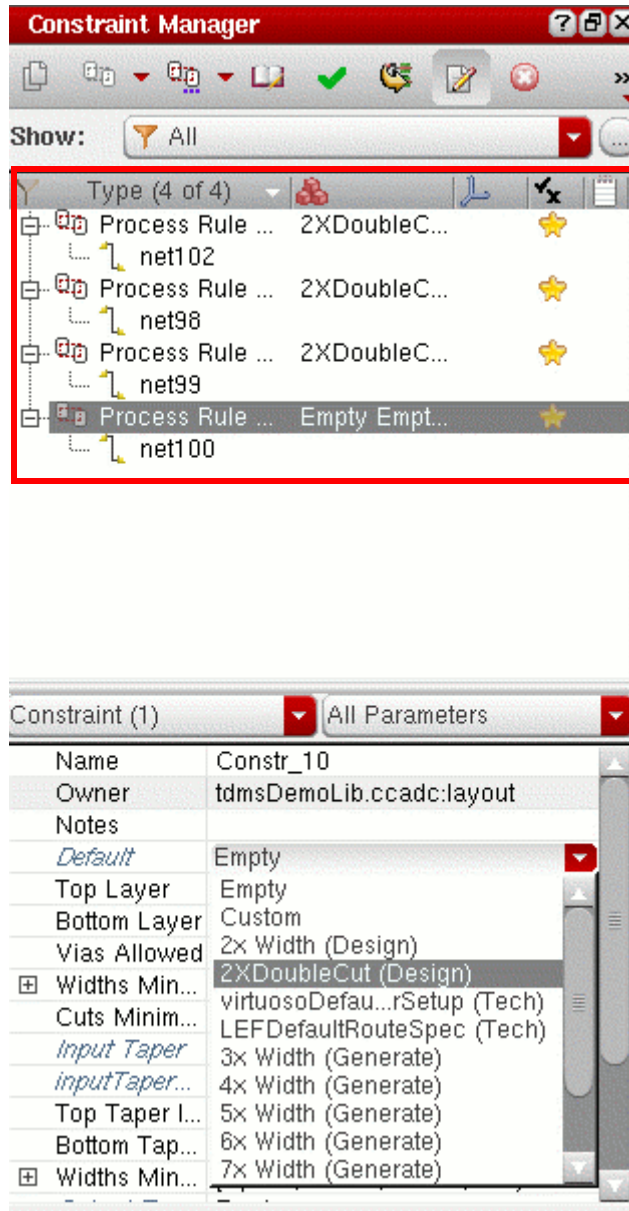


6. Repeat steps 3 through 5 for each net to which you want to apply these constraints.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

This lets you apply the same process rule overrides to different objects without having to re-enter the same process rules manually every time.



Note: The changes you make in the Process Rule Editor can affect the value shown in the *Default* field in the *Constraint Manager*.

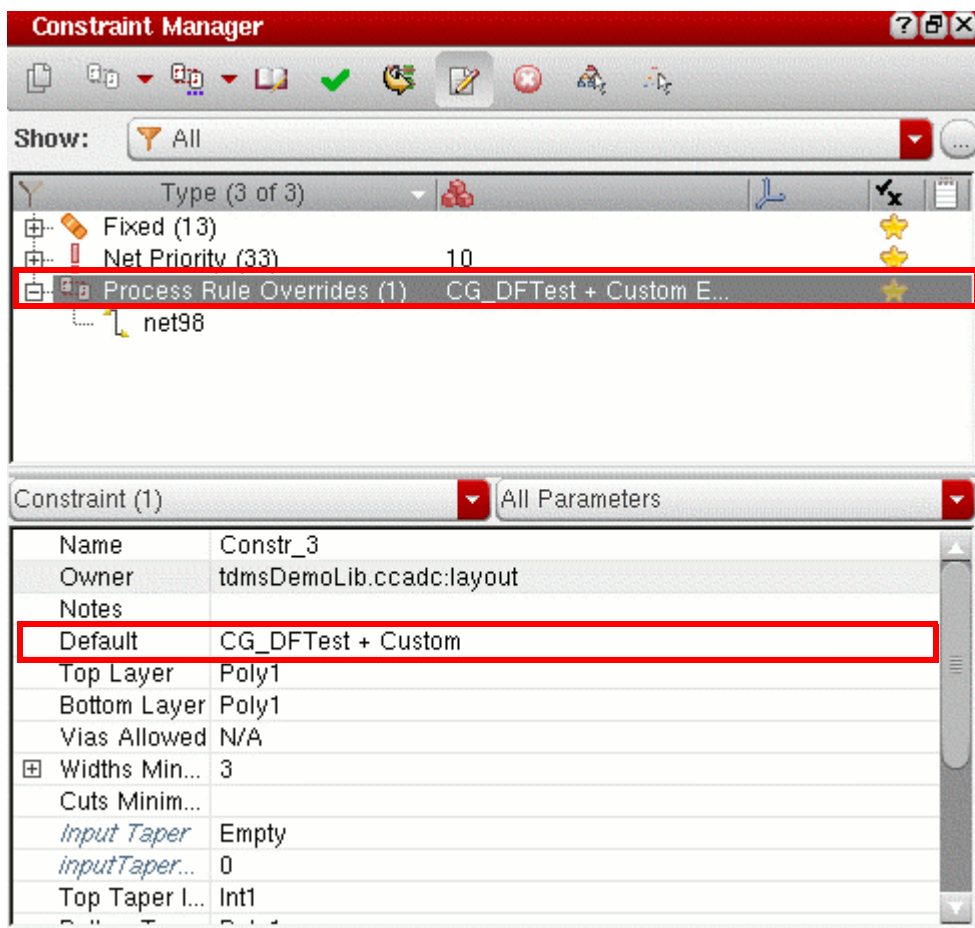
For example, assume the *Default* field in the *Constraint Manager* shows *Empty*. You open the Process Rule Editor and apply the *2xDoubleCut* constraint group. The *Default* field in the *Constraint Manager* now updates to show *2xDoubleCut (Design)*.

■ If the Process Rule Editor is empty, the *Default* field shows *Empty*.

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

- If the Process Rule Editor contains a single group, the *Default* field always shows the name of that group.
- If you apply some process rules to the Default Constraint Group in the Object scope, the *Default* field in the Process Rule Editor shows *Custom*. If you then apply a constraint group to the *Default* constraint group, the value of the *Default* field changes to a constraint group string. The string includes the name of the new constraint you added. In addition, it displays *Custom* to indicate you have also added some custom Process Rules to the Default constraint group. The constraint string is displayed by both, the Constraint Manager and the Constraint Editor.



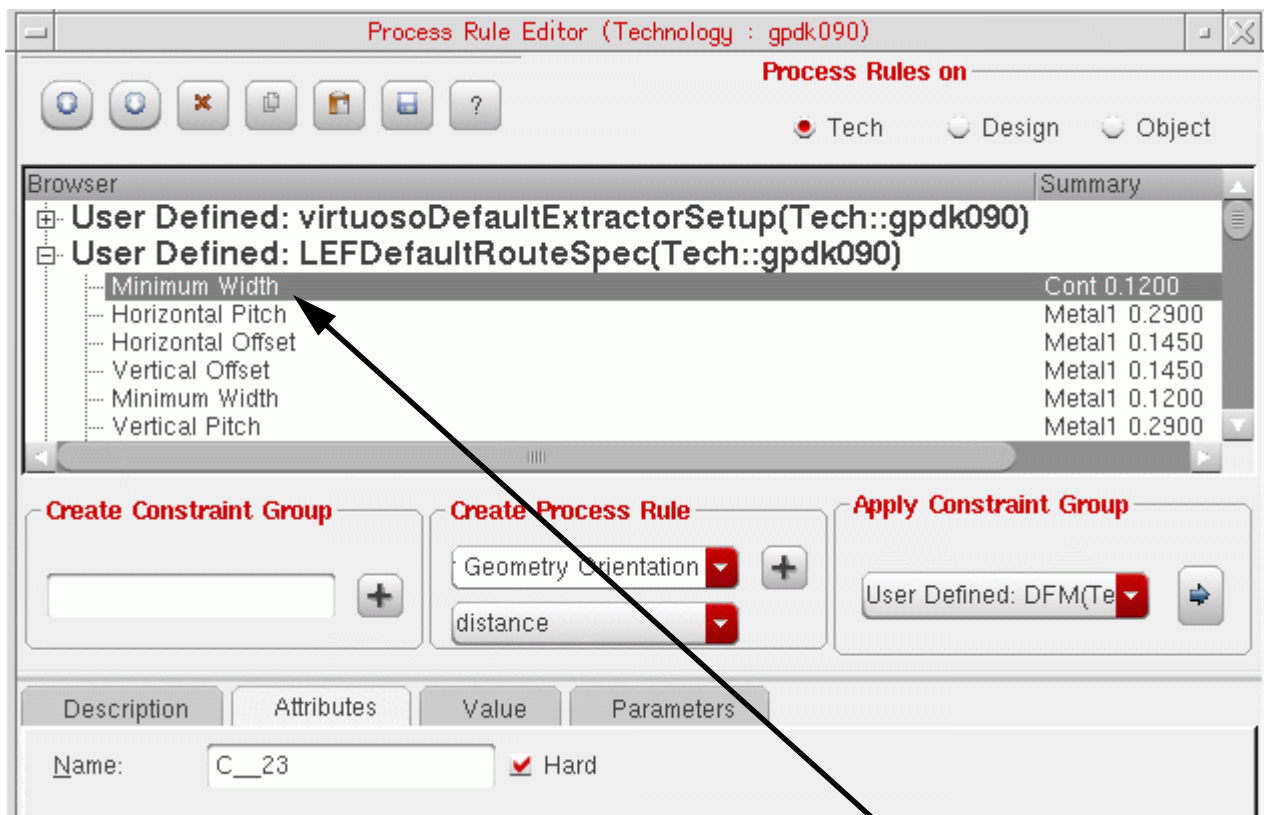
- When you try to change the setting of the *Default* field from *Custom* to a different value, you see a warning that doing so will replace all the custom edits you have made. This prevents you from inadvertently replacing a large amount of custom work with a single reference to a constraint group.

Removing a Constraint Group or Process Rule

You can remove a constraint group or an individual process rule from the Process Rule Editor to prevent it from being applied to the current design or object. The behavior is different depending on what you are removing.

If the selected object is:

- **A process rule**, the rule is deleted from the technology database.

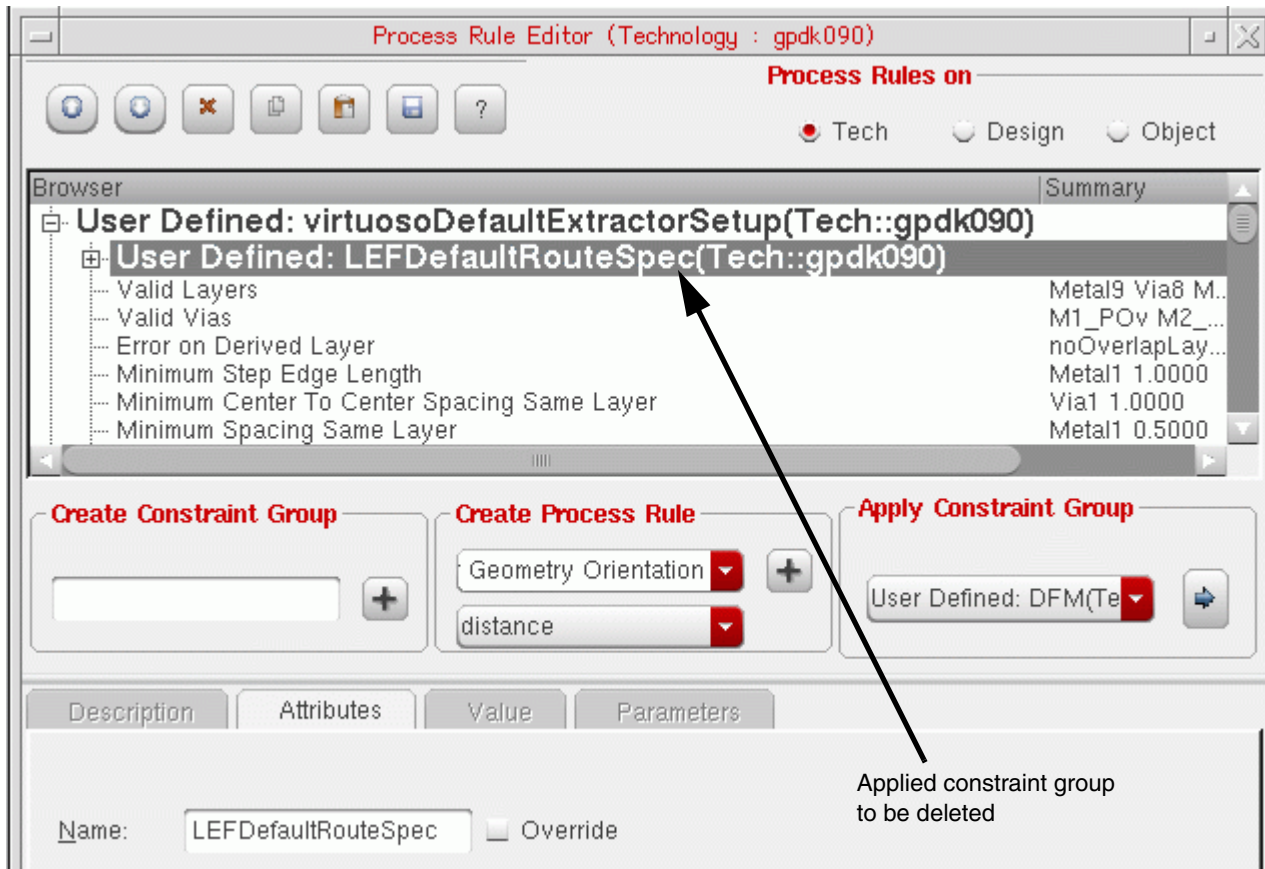


Process rule to be deleted from the technology database

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

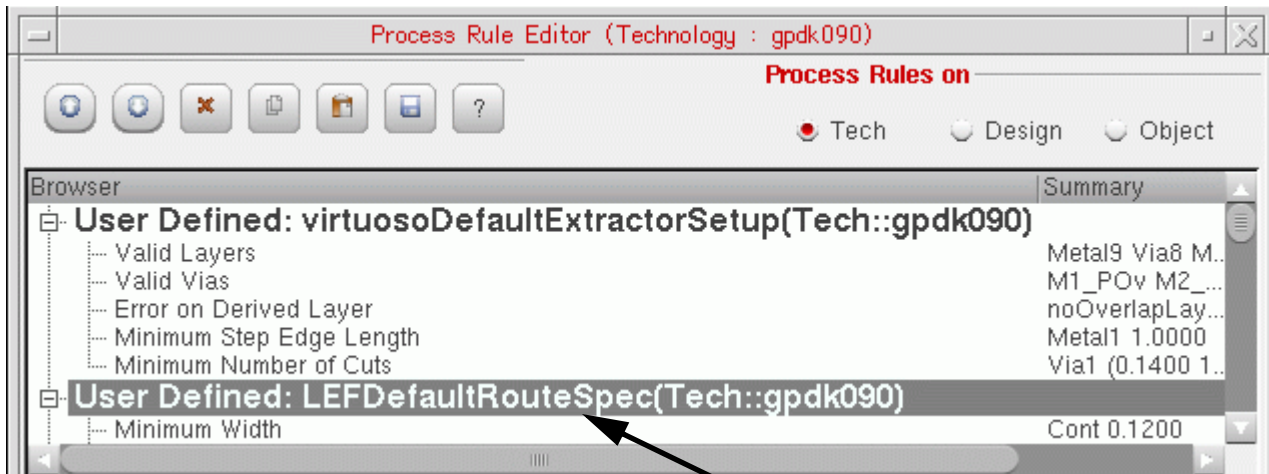
- **A constraint group applied to another constraint group**, the reference is removed from the browser pane but the constraint group is not deleted from the technology database.



Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

- ❑ After the constraint group is removed, it still appears in the technology scope.

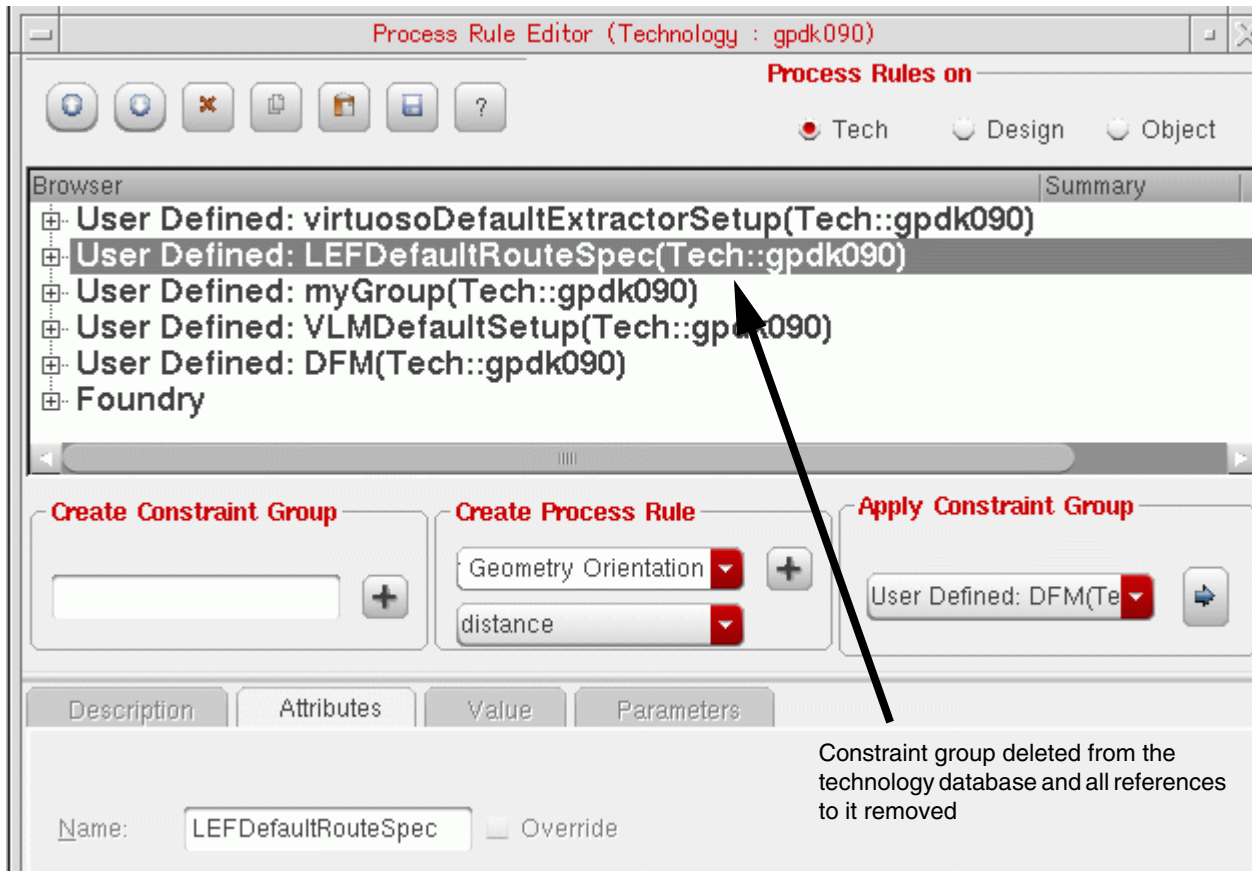


Constraint group still appears in
the technology scope

Virtuoso Unified Custom Constraints User Guide

The Process Rule Editor

- A constraint group definition on the main branch of the browser pane, the constraint group is deleted from the technology database, is removed from the main branch, and removed from any constraint group that references it.



To remove a constraint group or process rule from the browser pane:

1. Select the constraint group or process rule you want to remove.
2. Do one of the following:
 - ❑ Click the right mouse button and choose *Remove*.
 - ❑ Click the *Remove Selected* button in the toolbar.



Saving Changes and Closing the Process Rule Editor

To save the changes you have made at any time:

- Click the *Save Technology* button in the Process Rule Editor toolbar.



The changes are saved but the Process Rule Editor is not closed. Use this to save your work periodically as you define process rules.

To close the Process Rule Editor:

- Click the *Exit* button in the bottom right corner of the Process Rule Editor window.
Any unsaved changes are saved and the Process Rule Editor window closes.

Related Topics

For a list of process rules supported by the Process Rule Editor, see

- [Process Rules Supported by the Process Rule Editor](#)

For more information on the Process Rule Editor graphical user interface, see

- [Process Rule Editor Graphical User Interface](#)

Default Constraint Types

A constraint type is a formal definition of a design annotation that describes required electrical or physical relations between a set of design objects (constraint members) through a set of pre-defined constraint parameters.

Note: In addition to the default system constraint types discussed in this appendix, you can create your own custom constraint types by using the UI configuration file, `config.xml`. You can also use this file to override definitions of the existing custom constraint types and extend definitions of any system constraint type. For detailed information, refer to [Customizing Constraint Types Using a Configuration File](#) in the *Virtuoso Unified Custom Constraints Configuration Guide*.

This appendix contains information on the following:

- [Constraints in Virtuoso Schematic and Layout Editing Environment](#)
- [Constraints in Virtuoso Parasitic Aware Design Environment](#)
- [Axis Parameter](#)
- [Widths Minimum Parameter](#)
- [Spacing Parameter](#)
- [Parameter Types](#)
- [Support of Shapes](#)

Constraints in Virtuoso Schematic and Layout Editing Environment

Schematic Constraint Manager Toolbar		Layout Constraint Manager Toolbar	
<i>Electrical</i> menu	<u>Max Resistance</u>	<i>Electrical</i> menu	<u>Max Resistance</u>
	<u>Max Voltage Drop</u>		<u>Max Voltage Drop</u>
	<u>Max Capacitance</u>		<u>Max Capacitance</u>
	<u>Matched Capacitance</u>		<u>Matched Capacitance</u>
	<u>Max Coupling Capacitance</u>		<u>Max Coupling Capacitance</u>
	<u>High Precision C Extraction</u>		<u>High Precision C Extraction</u>
	<u>High Precision R Extraction</u>		<u>High Precision R Extraction</u>
	<u>Voltage Synced Net</u>		<u>Delta Voltage</u>
	<u>Correlation</u>		<u>Matched Parameters</u>
	<u>Matched Parameters</u>		<u>Matched LDE Parameters</u>
	<u>Matched LDE Parameters</u>		

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Schematic Constraint Manager Toolbar		Layout Constraint Manager Toolbar	
Placement menu	<u>Alignment</u>	Placement menu	<u>Alignment</u>
	<u>Cell Boundary</u>		<u>Cell Boundary</u>
	<u>Rail</u>		<u>Rail</u>
	<u>Cluster</u>		<u>Cluster</u>
	<u>Cluster Boundary</u>		<u>Cluster Boundary</u>
	<u>Disable Permutation</u>		<u>Disable Permutation</u>
	<u>Distance</u>		<u>Distance</u>
	<u>Guard Ring</u>		<u>Guard Ring</u>
	<u>Matched Orientation</u>		<u>Matched Orientation</u>
	<u>Modgen</u>		<u>Fixed</u>
	<u>Orientation</u>		<u>Locked</u>
	<u>Symmetry</u>		<u>Modgen</u>
	<u>Placement Path</u>		<u>Orientation</u>
			<u>Symmetry</u>
			<u>Placement Path</u>
	<u>Area Pin Group Guide</u>		
	<u>Edge Pin Group Guide</u>		
Routing menu	<u>Bus</u>	Routing menu	<u>Bus</u>
	<u>Diff Pair</u>		<u>Diff Pair</u>
	<u>Matched Length</u>		<u>Fixed</u>
	<u>Net Class</u>		<u>Locked</u>
	<u>Net Class Group</u>		<u>Matched Length</u>
	<u>Net Class Hier Group</u>		<u>Net Class</u>
	<u>Net Priority</u>		<u>Net Class Group</u>
	<u>Shielding</u>		<u>Net Class Hier Group</u>
	<u>Symmetry</u>		<u>Net Priority</u>
	<u>Process Rule Overrides</u>		<u>Shielding</u>



The Constraint Manager does not allow certain infeasible combinations of constraint and constraint parameters. For example, an orientation constraint with allowed orientation parameter set to R0 on two instances that have a Symmetry constraint with the Mirror parameter set to true. It does not create such constraints and generates a warning message in the CIW.

Alignment

Definition

The Alignment constraint defines alignment between members of the constraint. Its most important parameters are *side* and *use*, which define the reference for the alignment.

- Parameter *side* defines what the reference point for the alignment is. In case parameter *side* has the value *edgeNumber*, the parameter *edge* has an effect and must be specified on the member.

Note: Currently, the *side* parameter on individual constraint members are ignored, except by the *Virtuoso Floorplanner*.

- Parameter *use* specifies whether the member's boundary or one of its layers shall be used as reference for alignment. In case parameter *use* is set to *layer*, parameter *layer* has an effect and must be specified.

The parameter *anchor* may be set to the value *true* for zero or one members. If a member's *anchor* parameter is set to *true*, then it is defined as the "leader" in the member set. The *preserveOrder* parameter keeps the members in the same spacial order along the axis of alignment.

The alignment constraint accepts an Axis Parameter. If no axis is specified, the constraint uses a default axis. Two alignment constraints that have a default axis do not always or necessarily share the same axis.

Note: Only VLM supports shapes as members. All other applications, such as Analog Placer, Constraint Aware Editing, Digital Placer, and Floorplanning will ignore them.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Parameters

In addition to the parameters *Name*, *Enabled*, and *Notes*, you can set the following parameters:

Note: There is no way to disable the status value in schematic. Users should ignore it, except for Matched Parameter.

■ **Axis** (see [Axis Parameter](#))

For alignment constraints, the only information read off of the axis is the *direction* of the axis. The alignment constraint does not look at the *realizedCoordinate*, *coordinate* or *axisLocation* sub-parameters.

■ **Side**

Type: enum

Default: “left” (when vertical axis selected), “top” (when horizontal axis selected)

Valid Values: “top”, “center”, “bottom”, “right”, “origin”, “edgeNumber”, “orderOnly” (for vertical axis) “left”, “center”, “origin”, “edgeNumber”, “orderOnly” (for horizontal axis)

The “center” parameter value specifies that members be aligned using their centers, and not their edges. If a constraint has both pin and PR boundary members, the “center” value is currently undefined. However, if the alignment constraint has pin members, but no PR boundary member, then the “center” value instructs that the pins should be aligned with regard to their centers. The side parameter value “edgeNumber” is only valid as a constraint parameter if the members are a polygonal (non-rectangular) PR boundary and pins. In the latter case, the side parameter of the pins must be set to “edgeNumber” and the value of the edge parameter of the pins must be set to the corresponding edge number of the polygonal PR Boundary for alignment.

For more information on the use of “edgeNumber”, and its use in the *Virtuoso Custom Digital Placer*, see the *Placement Constraints* section in the [Introduction](#) chapter.

On Constraint: Y

On Member: Y

Display: Y

User Units: N

■ **PreserveOrder**

Type: boolean

Default: false

On Constraint: Y

On Member: N

Display: Y

User Units: N

If *PreserveOrder* is set to *true*, then the Analog Placer will place the pins in the order that they appear as members in the alignment constraint. If you want a different order to be preserved you must change the order that the pin members appear in the constraint.

Note: Value cannot be false when side = orderOnly

■ **pitchReference**

Type: enum

Default: "center"

Valid Values: "edge" and "center"

On Constraint: Y

On Member: N

Display: Y

User Units: N

■ **pitch**

Type: integer

Default: (not set)/-1

Valid Values: positive

On Constraint: Y

On Member: N

Display: Y

User Units: Y

Note: The parameter pitch is a center-to-center distance.

■ **use**

Type: enumerated

Default: “boundary”

Valid Values: “boundary” “layer”

On Constraint: Y

On Member: Y

Display: Y

User Units: N

Note: Alignment should use a layer (with layer name) for alignment instead of the boundary as the same side applies.

■ **layerName**

Type: integer

Default: 0

Valid Values: layer names

On Constraint: Y

On Member: Y

Display: Y

User Units: N

The *layerName* parameter has a drop-down list box in the *Constraint Manager* for its field value. This menu contains a list of all possible layer selections that can be used to align alignment constraint members. Currently, this drop-down list box is populated using all the layers that are defined in the technology file. This includes regular and derived layers.

If the *use* parameter is specified to be “layer”, then the *layerName* parameter will be referenced by Constraint Aware Editing (in Virtuoso Layout Suite XL). If that is the case, the alignment will occur using the specified layer’s bounding box, as opposed to the figure’s bounding box.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Members

Two or more instances, modgen constraints or figGroups, or regular figGroups, pins, terminals, shapes. One or more pins can be members along with a boundary.

Alignment constraints that have **pins** as members, should align to the PR Boundary edge specified by the *side* parameter on the constraint (left, right, top, bottom) *only* if a PR Boundary is also a member of the constraint. If an alignment constraint contains pins as members, but has no PR Boundary object as a member, the pins should align anywhere on the canvas, but not snap to PR Boundary edge.

Note: Each individual member of an alignment constraint can have a different set of parameters, for example, one member can be placed ABOVE and one member BELOW the horizontal axis, using the *side* parameter of each member. If the value is specified on the member, then that value will be *user*, otherwise the value set on the constraint will be inherited for all members that do not have an explicit value. Alignment algorithms will respect the individual values of each member.

Member order is important when the parameter *preserveOrder* is set to `true`.

Supporting Applications

- Virtuoso Custom Digital Placer (uses all parameters except *use*, *layerName*, and *anchor*)
- Virtuoso Floorplanner (uses all parameters except *use* and *layerName*)
- Virtuoso Analog Auto Placer (all parameters except *pitch* and *edge*)
- Virtuoso Layout Suite XL / Constraint Aware Editing (all parameters except *preserveOrder(pins)*, *pitch*, *edge*, and *anchor*)

Note: *preserveOrder* is only supported when it is applicable to pins. That is, when the alignment constraint has pin members.

Area Pin Group Guide

Definition The Area Pin Group Guide constraint defines the placement of pin members within the guide regions defined by its area boundary members.

Parameters In addition to the parameters *Name*, *Enabled*, and *Notes*, you can set the following parameters:

■ Default Group (Constraint Group)

The *Default Group* contains the following parameters: *Valid Layers*, *Widths Minimum*, *Spacing*, *layerPriorityOrder*, *preserveOrder*, *spreadPins*, and *pinGroupType*.

- ❑ *Valid Layers* defines the layers that can be used when placing the member pins.
- ❑ *Widths Minimum* defines the default minimum width for member pins in the pin group guide. Width is defined per layer in user units.
- ❑ *Spacing* defines the default minimum spacing for the pin group guide. Spacing is defined per layer in user units.
- ❑ *layerPriorityOrder* defines whether the *Valid Layers* are listed in the order of preference.

Type: Boolean

Default: false

- ❑ *preserveOrder* defines whether the member pins are placed in the same order as the members.

Type: Boolean

Default: false

- ❑ *spreadPins* defines whether the member pins are evenly spaced.

Type: Boolean

Default: false

- ❑ *pinGroupType* defines whether the group is inclusive or exclusive. It specifies if other non-member pins are allowed in the regions specified by the group's area boundary members.

Type: enum

Default: AllowForeignPins

Values: This parameter accepts one of the following values:

- *AllowForeignPins* means other pins are allowed in the whole region.
- *ExcludeOnAllLayers* means exclude other pins between members on all layers.
- *ExcludeOnPinLayers* means exclude other pins between members on pin layers.
- *ExcludeForeignPins* means exclude other pins from whole region on all layers.

■ Within Group (Constraint Group)

The *Within Group* contains the *Spacing* parameter. The Within Group parameters are used to enter process rules between pin members in the group.

- ❑ *Spacing* defines the minimum spacing between a pin and other member pins. Spacing is defined per layer in user units.

■ Group to Outside (Constraint Group)

The *Group to Outside* contains the *Keep Out Spacing* parameter. The Group to Outside parameters are used to enter process rules between pin members and other pins in the design.

- ❑ *Keep Out Spacing* defines the minimum spacing between the pin group guide members and other pins. Spacing is defined per layer in user units.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Members

The *Area Pin Group Guide* members are one or more pins, plus one or more area boundaries. The area boundary members define the regions in which the pin members can be placed.

Supporting Applications

■ Virtuoso Floorplanner

- Pin Group Guide commands to Create/Edit and Compare

For detailed information, refer to the *[Creating Pin Group Guides](#)* chapter in the *Virtuoso Floorplanner User Guide*.

- Pin Planning and Pin Optimization

For detailed information, refer to the following sections of the *Pin Placement* chapter in the *Virtuoso Floorplanner User Guide*:

- *[Pin Planner](#)* section
- *[Pin Optimization](#)* section

■ Innovus™ Digital Implementation System (Innovus)

Note: This constraint type is interoperable with Innovus and validated by PVS-CV.

Bus

Definition The Bus constraint is used to create new groups of nets that routers can then use as buses. The nets constrained will therefore be routed physically as a bus in the layout, following the same topology and spacing.

The Process Rule Editor can be used to apply process constraints on the spacing between the bussed nets, as well as the spacing between the bus and all other nets.

Parameters In addition to the parameters *Status*, *History*, *Name*, *Enabled*, and *Notes*, you can set the following parameters:

■ **Default Group (Constraint Group)**

The *Default Group* category contains the following parameters: *Top Layer*, *Bottom Layer*, *Purposes*, *Vias Allowed*, *Widths Minimum*, *Cuts Minimum*, *Spacing*.

The above parameters are used to reflect the contents of a Bus default constraint group. Constraints may be placed here to locally override foundry rules obtained through standard lookup. Adding a constraint parameter to a Bus's default constraint group is equivalent to adding the same constraint in each net member's default constraint group. If there are similar constraints in the default constraint group of the net and the bus, then the net's default constraint group parameter settings takes the precedence. A typical use here would be to route a set of nets with a specific wider width or a net with a specific number of via cuts, or to modify the valid layers for all nets in a bus.

Note: The *Default Group* category is available only when the members of a Bus constraint are nets (that is, not a nested net class).

In addition to the parameters listed above, the *Default Group* category has the *Wire Types* parameter that can be used to define a set of wire types for each layer. For more information, see Assigning a Wire Type to a Net.

For more information on the Default Group, see [The Process Rule Editor](#).

For more information on customizing constraint groups, see [Customizing a Constraint Group](#).

■ Within Group (Constraint Group)

The *Within Group* category contains the following parameter: *Spacing*.

The *Spacing* parameter is used to enter process rules between net members in a Bus constraint.

Note: The *Within Group* category is available only when the members of a Bus are nets (that is, not a nested net class).

For more information on the Within Group, see [The Process Rule Editor](#).

■ Group to Outside Group (Constraint Group)

The *Group to Outside Group* category contains the following parameters: *Spacing*, *Tolerance*, *hierarchicalScope*.

The parameters available in the *Group to Outside Group* category are used to specify spacing for the routed bus to everything else in the design.

The *Group to Outside Group* category can be used to enter process rules between nets in a bus constraint and those nets that are not part of that constraint.

For more information on Group to Outside Group, see [The Process Rule Editor](#).

Members

Two or more nets.

Supporting Applications

- Constraint Aware Editing (Create Wire)
- Virtuoso Space-based Router

Cell Boundary

Definition The Cell Boundary constraint defines implicitly the aspect ratio for floorplanning tools using the following parameters:

- *minWidth*
- *maxWidth*
- *minHeight*
- *maxHeight*
- *mode*

By default, all the parameters have a range from zero to positive infinite.

Parameters In addition to the parameters *Name*, *Enabled*, and *Notes*, you can set the following parameters:

- **minWidth**

Type: float

Default: 0 [Database Units]

Valid Values: positive

On Constraint: Y

On Member: N

Display: Y

User Units: Y

■ **maxWidth**

Type: float

Default: 0 [Database Units]

Valid Values: positive

On Constraint: Y

On Member: N

Display: Y

User Units: Y

■ **minHeight**

Type: float

Default: 0 [Database Units]

Valid Values: positive

On Constraint: Y

On Member: N

Display: Y

User Units: Y

■ **maxHeight**

Type: float

Default: 0 [Database Units]

Valid Values: positive

On Constraint: Y

On Member: N

Display: Y

User Units: Y

■ **mode**

The *mode* parameter allows you to specify feasible combinations as a range (min/max):

Width, Height, Aspect Ratio, Area Utilization, and Area.

- ❑ `minWidth` and `maxWidth` only shows up when mode contains Width.

Default for `minWidth` is 0

Default for `maxWidth` is 50

- ❑ `minHeight` and `maxHeight` only shows up when mode contains Height.

Default for `minHeight` is 0

Default for `maxHeight` is 50

- ❑ `minAspectRatio` and `maxAspectRatio` only shows up when mode contains AspectRatio.

Type: float

User Unit: N

Default for `minAspectRatio` is 0.8

Default for `maxAspectRatio` is 1.2

- ❑ `minUtilization` and `maxUtilization` only shows up when mode contains AreaUtilization. The percentage of devices area inside total boundary area is 50, which means 50%.

Type: float

User Unit: N

Default for `minUtilization` is 70

Default for `maxUtilization` is 80

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

- ❑ Area user desired area and does not have any min/max range.

User Unit: Y

Default value is 10000

The allowable parameter combinations are:

- ❑ *Width & Height*
- ❑ *Width & AreaUtilization*
- ❑ *Width & AspectRatio*
- ❑ *Width & Area*
- ❑ *Height & AreaUtilization*
- ❑ *Height & AspectRatio*
- ❑ *Height & Area*
- ❑ *AreaUtilization & AspectRatio*
- ❑ *AspectRatio & Area*

Members

A *Cell Boundary* constraint can be created without any selection having been made in the design canvas in both schematic and layout. The newly created constraint will be displayed in the *Constraint Manager* with one member, *PR Boundary*.

Note: The *Cell Boundary* constraint menu entry will be grayed out if you select a *PR Boundary* other than that belonging to the current cellview.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Supporting Applications

- Virtuoso Floorplanner
- Virtuoso Layout Suite XL (Constraint Aware Editing)
- Virtuoso Analog Auto Placer (*Partial Support*):
Supports all parameters of the Cell Boundary constraint, but only the following values of the *mode* parameter:
 - *Width & Height*
 - *Width & AspectRatio*
 - *Height & AspectRatio*
 - *Height & Area*
 - *Width & Area*
 - *AspectRatio & Area*

Cluster

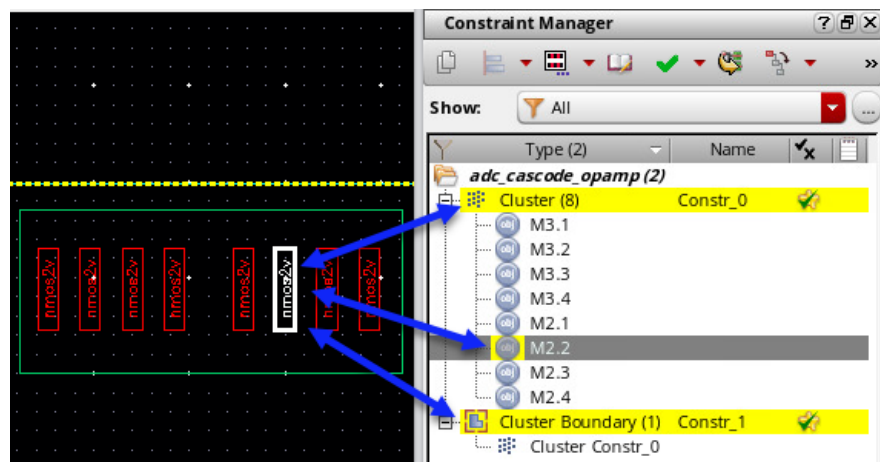
Definition

The Cluster constraint defines sets of instances and/or other clusters that exist in relatively close proximity. Both cluster selection and creation can be achieved using the *Constraint Manager* assistant.

Clusters can be created on a single schematic instance, hierarchical or otherwise, and successfully transferred to layout following the physConfig. To do this, use one of the following methods:

- At an instance level, do one of the following:
 - select the individual hierarchical instance in the canvas to create a Cluster constraint.
 - expand the hierarchical node in the *Navigator* and select the members to create a Cluster constraint.
- At a cell level, descend into the instance and select members to create a Cluster constraint. This can then be changed to a single occurrence if required (see [Moving a Constraint to the Current Cell](#)).

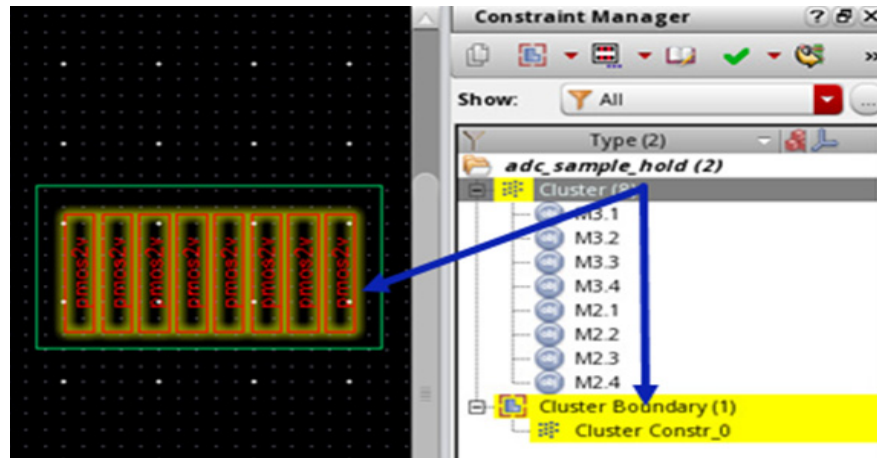
When you select a member of a Cluster constraint in the canvas, it gets cross-highlighted in the *Constraint Manager*. If the associated Cluster constraint is a member of a Cluster Boundary constraint, then that Cluster Boundary constraint also gets highlighted. The image below illustrates these scenarios from the Layout XL canvas.



Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

As illustrated below, when a cluster constraint is selected in the Constraint Manager assistant, the cluster boundary of which the constraint is a member gets cross-highlighted. The cluster boundary also gets haloed in the canvas.



The *Constraint Manager* assistant also supports the creation of hierarchical clusters, that is, clusters within clusters. For example, you can create a hierarchical cluster based upon the selection of existing clusters in the *Constraint Manager* assistant.

Note: The constraint checker cannot recognize certain types of user-defined Pcell guard ring instances and as a result, reports them as violating cluster constraints. To ensure that the constraint checker ignores such Pcell guard ring instances, set the `caelgnoreInCluster` property on the specific guard ring instance or its supermaster (to ignore all instances of that supermaster).

Parameters

Name, *Enabled*, and *Notes* parameter information can be set.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Members

One or more instances, modgen constraints or figGroups, regular figGroups (equivalent to adding the individual members), and/or other clusters. Membership follows oaCluster rules and the cluster tree must be a partition (one member in one cluster at most).

Note: In the front-end, Cluster constraints can only be created on scalar instances and vector bit instances, not on vector instances. This is because scalar and vector bit instances accurately describe the layout objects and design intent, while also being natively supported in the database.

Clusters that are selected in the Constraint Manager assistant should form part of the selected set. This is important in allowing constraints to be created containing hierarchical clusters.

The *Constraint Manager* will also consider any clusters currently selected to be part of the selected set when a Guard Ring constraint is created. This allows clusters to be members of Guard Ring constraints. For example:

1. Select existing Cluster constraint in *Constraint Manager*.
2. Use the `Shift` key to also select the net that is to be connected to the guard ring (for example, a wire of this net in the schematic).
3. Select the *Guard Ring* option from the *Constraint Generator* drop-down list box.
4. Select the net member in the *Constraint Manager*.
5. Edit the newly generated Guard Ring constraint to enter a valid MPP name that is to be used by the guard ring.

A Cluster constraint can also be a *constraint container*. That is, it can contain its own members and also be used as a member itself for other constraints.

Supporting Applications

- Virtuoso Custom Digital Placer
- Constraint Aware Editing
- Virtuoso Floorplanner

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Example: How to Add Cluster Members

The following example details how to include additional members on a cluster constraint using the `ciConUpdateMembers` API:

```
procedure(doClusterBoundaries(cv cache name align)
  let((conname1 conname2)
    printf("name in doClusterBoundariess is %L\n" name)
    if(align then
      if((conname1 = ciConFind(cache name)) then
        printf("adding in doClusterBoundariess is %L\n" align)

        ciConUpdateMembers(conname1 append(align foreach(mapcar x
ciConListMembers(conname1) list(car(x) cadr(x)))))

      else
        printf("creating in doClusterBoundariess is %L\n" align)
        conname1 = ciConCreate(cache 'cluster
          ?members align
          ?name name
        )
        ciConCreate(cache 'clusterBoundaryDef
          ?members list(list(name 'cluster))
          ?params list(list("boundary" x=getBoundary(cv "upper")))
list("flexibleFlag" 1))
        )
      )
    )
    printf("names = %L\n" ciConListMembers(conname1))
  )
)
```

Virtuoso Unified Custom Constraints User Guide

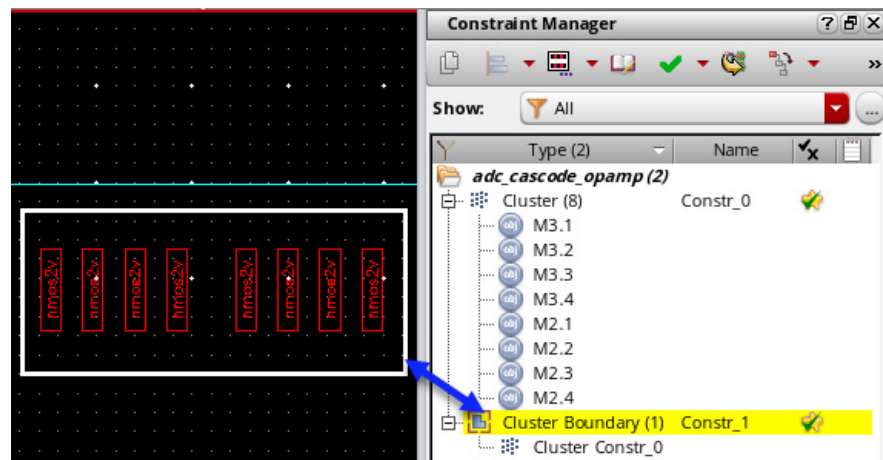
Default Constraint Types

Cluster Boundary

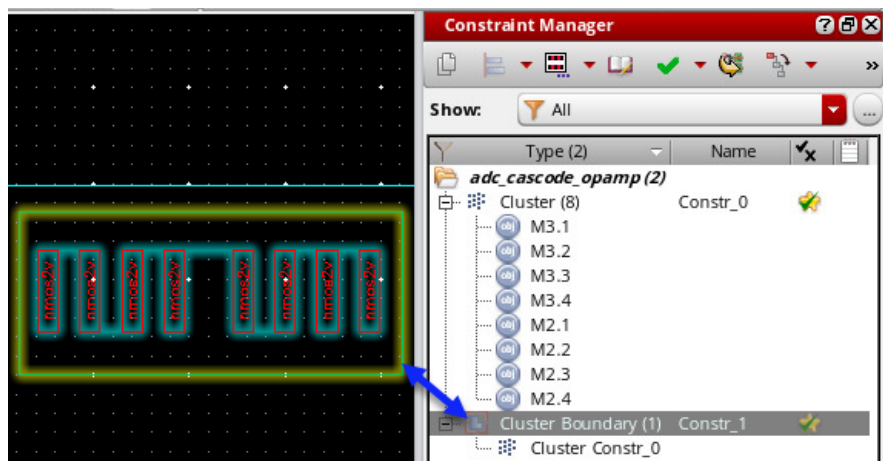
Definition

The Cluster Boundary constraint represents a boundary which its member cluster must respect.

When you select a cluster boundary in the canvas, the corresponding Cluster Boundary constraint gets highlighted in the Constraint Manager assistant, as shown below.



Similarly, on selecting a Cluster Boundary constraint in the Constraint Manager assistant, the associated cluster boundary and all members related to the cluster get haloed in the canvas, as shown below.



Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Parameters

In addition to the parameters *Name*, *Enabled*, and *Notes*, you can set the following parameters:

■ **flexibleFlag**

Type: Boolean

Default: 0

Valid Values: (0, 1)

On Constraint: Y

On Member: N

Display: Y

User Units: N

■ **boundary**

Type: float(4) - 4 indicating the number of float types

Default: (0 0) (0 0.001) (0.001 0.001) (0.001 0)

On Constraint: Y

On Member: N

Display: Y

User Units: N

Members

One cluster

Supporting Applications

- Virtuoso Custom Digital Placer
- Virtuoso Floorplanner
- Constraint Aware Editing

Correlation

Definition	The Correlation constraint is used to define the coefficient of correlation between two devices for Monte Carlo simulation.
Parameters	In addition to the parameters <i>Name</i>, <i>Enabled</i>, and <i>Notes</i>, you can set the following parameters: ■ Coefficient <i>Type:</i> float <i>Default:</i> 0.0 <i>Valid Values:</i> in [-1.0 1.0] <i>On Constraint:</i> Y <i>On Member:</i> N <i>Display:</i> Y <i>User Units:</i> N ■ correlatedParameters <i>Type:</i> string1DTable <i>Valid Values:</i> space separated list of names <i>On Constraint:</i> Y <i>On Member:</i> N <i>Display:</i> Y <i>User Units:</i> N
Members	Two or more unique instances. These instances cannot be used in another Correlation constraint. Any instance can be used in any number of Correlation constraints.
Supporting Applications	■ Monte Carlo Analysis

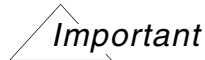
Diff Pair

Definition

The Diff Pair constraint defines a specialty routing relationship between two nets. The nets should be routed together in order to minimize variations from spacial dependent effects.

For more information, see [The Process Rule Editor](#).

Parameters



See also [Diff Pair Parameters and the Process Rule Editor](#) for additional Diff Pair parameter information.

In addition to the parameters *Name*, *Enabled*, and *Notes*, you can set the following parameters:

■ **Default Group (Constraint Group)**

Which contains the following parameters:

Top Layer, *Bottom Layer*, *Purposes*, *Vias Allowed*, *Widths Minimum*, *Cuts Minimum*, *Spacing*.

These constraint group parameters include information on the layers to use for Diff Pair. Parameters correspond to a valid layers constraint in the PRE. The layers parameters should always contain a contiguous set of layers to use for this net (/net group). The router should route using only these layers. Additionally, the spacing (for each layer) represents the min spacing default constraint group for Diff Pair.

In addition to the parameters listed above, *Default Group* has the *Wire Types* parameter that can be used to define a set of wire types for each layer. For more information, see [Assigning a Wire Type to a Net](#).

For more information on Default Group, see [The Process Rule Editor](#).

■ Within Group (Constraint Group)

The *Within Group* parameters help to define the process rules for the net members in a Diff Pair constraint. These parameters include *Spacing* and *Tolerance*. For more information on the *Within Group* parameters, see [The Process Rule Editor](#).

- ❑ The *Tolerance* parameter describes the percentage allowed deviation from specification when implementing a given constraint. This parameter applies only to symmetry (and other constraints) on nets, for the purpose of routing.
- ❑ In addition to the *Spacing* and *Tolerance* parameters, you can define the *Same Mask* parameter to ensure all shapes on a given layer for a net are on the same mask. This parameter allows you to specify the color of the nets in the net class. This can also have a hard or a soft attribute.

Note: The *Same Mask* parameter is equivalent to using the `msSameMask` constraint. If you reset the value of this constraint, the color of the nets should be reset accordingly.

- ❑ Optionally, in addition to the *Same Mask* parameter, you can set the *Mask Name* parameter to ensure that the shapes will be on the specified color mask for each routing layer for the nets in the Symmetry constraint.

For more information, refer to [Specifying the Color Mask for Critical Nets in Schematic](#).

■ Group to Outside Group (Constraint Group)

Which contains the following parameters:

Spacing, *Tolerance*, *hierarchicalScope*.

The Group to Outside Group constraint group is used to enter process rules between nets in a Diff Pair constraint and those nets that are not part of that constraint.

For more information on Group to Outside Group, see [The Process Rule Editor](#).

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Members

Two nets only.

A given net may belong to only 1 specialty routing constraint (for example, [Symmetry](#), Diff Pair, or [Matched Length](#)).

Note: A Diff Pair constraint can also be a *constraint container*. That is, it can contain its own members and also be used as a member itself for other constraints.

Supporting Applications

- Constraint Aware Editing (Create Wire)
- Virtuoso Space-based Router
- Innovus System

Diff Pair Parameters and the Process Rule Editor

The following Diff Pair constraint parameters allow to edit the process rules in [The Process Rule Editor](#). That is, changing the value of these parameters in the Constraint Manager assistant make equivalent changes in the PRE and conversely. Each parameter tooltip also provides information on where, in the constraint lookup, a value was found.

Diff Pair Parameter	PRE Equivalent Parameter Name	Notes
diffPair layers	<i>Valid Layers</i> on diffPair default constraint group	Specifies the min and max of a range of valid layers that routers should use on a net when tapering. If the database does not currently contain a sequential list of layers then these fields will be blank. Note: In the UI, enter a top and bottom layer entry to the specify the range.
gap spacing	<i>Gap Spacing</i> in diffPair <i>Within Group</i>	Specifies the spacing to use per layer between the diffPair wires.

Disable Permutation

Definition

The Disable Permutation constraint allows you to stop permutation on a device without having write access to the schematic. It can also provide additional visibility when permutation is important.

Note:

- ❑ The Disable Permutation constraint will override the `permuteRule`. Permutation constraints which are not met are ignored and do not disable permutation.
- ❑ One constraint is applied per instance and is used to check that the schematic and layout permutations are the same. This can be used to preserve current direction.
- ❑ The instance's `permuteRule` will return `nil` when the constraint is active, and met.
- ❑ Global `permute` operates as normal when the constraint is deleted.
- ❑ Permute constraints apply throughout the flow.
- ❑ The Disable Permutation constraint has been created to keep the connectivity of a device in the layout the same as it is in the schematic.

Parameters

Name, *Enabled*, and *Notes* parameter information can be set.

Members

One instance only.

Note: It can be useful to apply the Disable Permutation constraint to any instance with connectivity and a `permute` rule. Typically the Disable Permutation constraint is used with transistors.

Supporting Applications

- Virtuoso Layout Suite XL (Constraint Aware Editing)

Note: The Disable Permutation constraint is also preserved in the Analog Auto Placer.

Distance

Definition

The Distance constraint defines the distance between two or more constraint members. The parameters *Min* and *Max* define the range for the distance. The parameter *reference* can be used as a constraint parameter or as a member parameter, and specifies the reference to be used for distance constraining, for example one of “edge”, “center” or “origin”.

When a Distance constraint has more than two members the Distance constraint applies between the first and any of the following objects. That is, each of the second and following objects must be within the specified range from the first object.

The Distance constraint accepts an Axis Parameter. Specifying an axis will restrict the constraint to one particular direction. If no axis is specified then the Distance constraint is bidirectional, applying in two dimensional space as a Manhattan distance.

The same pair of objects can be part of two Distance constraints as long as both specify an axis and the axes are orthogonal. Two constraints with the same members, but orthogonal axes, should be used to set different vertical and horizontal allowed distances.

See also: [Measuring a Distance Constraint When No Axis is Specified](#).

Note: Only VLM supports shapes as members. All other applications, such as Analog Placer, Constraint Aware Editing, Digital Placer, and Floorplanning will ignore them.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Parameters

In addition to the parameters *Name*, *Enabled*, and *Notes*, you can set the following parameters:

■ **Axis** (see [Axis Parameter](#))

■ **Min**

Type: float

Default: (not set) / 0

Valid Values: positive

On Constraint: Y

On Member: N

Display: Y

User Units: Y

■ **Max**

Type: float

Default: (not set) / INT_MAX

Valid Values: positive

On Constraint: Y

On Member: N

Display: Y

User Units: Y

Note: For the *Min* and *Max* parameters, see also [Example Uses of the Min and Max Distance Parameters](#).

■ **use**

Type: enumerated

Default: “boundary”

Valid Values: “boundary” “layer”

On Constraint: Y

On Member: Y

Display: Y

User Units: N

Note: The *use* parameter should use a layer (with layer name) for distance instead of the boundary as the same side applies.

■ **layerName**

Type: integer

Default: 0

Valid Values: layer names

On Constraint: Y

On Member: Y

Display: Y

User Units: N

The *layerName* parameter has a drop-down list box in the *Constraint Manager* for its field value. This list box contains a list of all possible layer selections that can be used to drive distant constraint members. Currently, this drop-down list box is populated using all the layers that are defined in the technology file. This includes regular and derived layers.

If the *use* parameter is specified to be “layer”, then the *layerName* parameter will be referenced by Constraint Aware Editing (in Virtuoso Layout Suite XL). If that is the case, the distance will occur using the specified layer’s bounding box, as opposed to the figure’s bounding box.

■ **reference**

Type: enum

Default: “edge”

Valid Values: “edge”, “center”, “origin”

On Constraint: Y

On Member: Y

Display: Y

User Units: N

Notes:

- min and max are distance range parameters, where the same values on both min and max indicates a fixed distance
- there is no ordered parameter to specify whether the order of objects should be preserved. An ordering constraint is required (distance: order only) to order devices.
- reference is one of “edge”, “center”, or “origin”. If reference is not set on members, the reference value for the constraint is used.
- When PRBoundary is a member, only “edge” is currently supported (by the Virtuoso Floorplanner). Parameter “edge” is also supported by Analog Auto Placer and Constraint Aware Editing.

Members

One or more instances, pins, shapes, modgen constraints or figGroups, or regular figGroups. PRBoundary can also be defined as a member, but this is only supported by the Virtuoso Floorplanner.

Member Order is important. When more than two objects are members, the distance between the first and any of the following objects has to be within the (min max) range for the constraint to be enforced.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Supporting Applications

- Virtuoso Custom Digital Placer
 - Virtuoso Floorplanner
 - *Virtuoso Analog Auto Placer
 - *Virtuoso Layout Suite XL (Constraint Aware Editing)
- * For target optimization only and not guaranteed to be met.

Example Uses of the Min and Max Distance Parameters

Both the *Min* and *Max* parameters display a default value of *(not set)* in the Constraint Manager, for *Min* that default equates to 0 and for *Max* it equates to `INT_MAX`.

The following examples show how you can also set one of these parameters without the need to set the other.

- No *Min* or *Max* values set: *Min*: (not set), *Max*: (not set)
Note: As mentioned, this equates to 0 (for *Min*) and `INT_MAX` (for *Max*).
- Only *Min* distance set: *Min*: 22.5, *Max*: (not set)
- Only *Max* distance set: *Min*: (not set), *Max*: 44.5
- A range of distance parameters set: *Min*: 2.5, *Max*: 22.0.

Measuring a Distance Constraint When No Axis is Specified

A Distance constraint can still be measured if only one Axis Parameter has been specified or if neither axis has been specified.

The figure below shows the following examples of this:

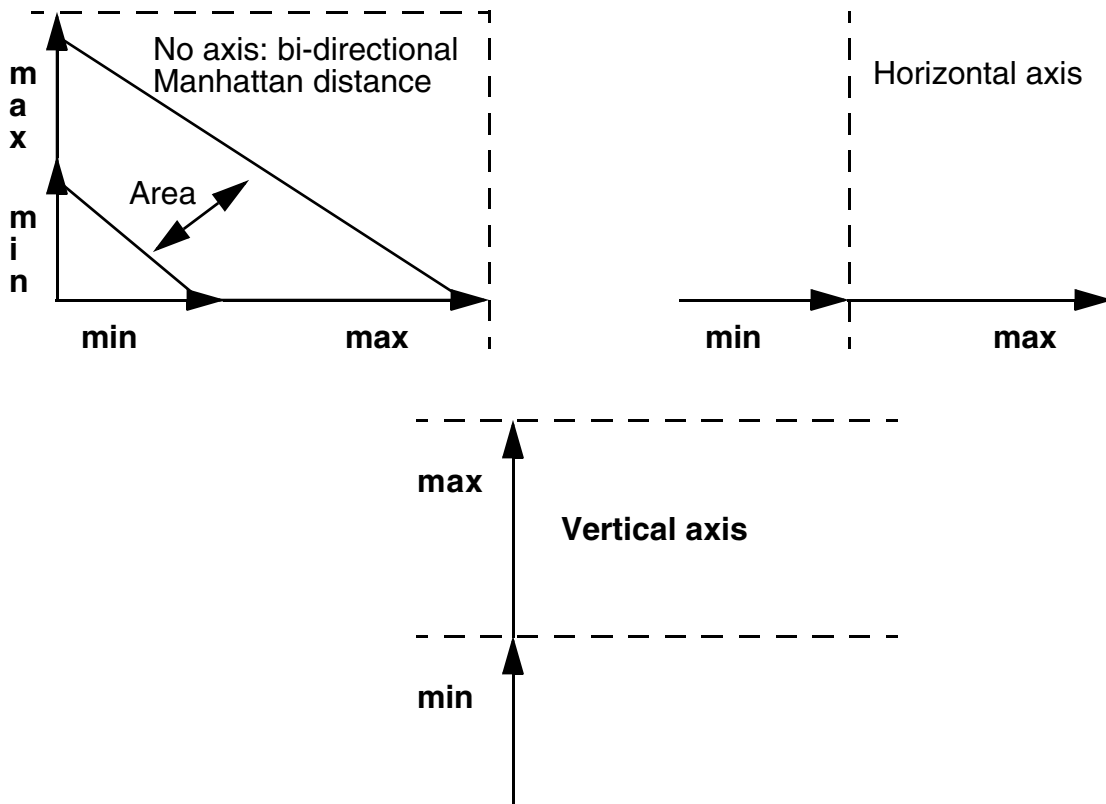
- In the top left, direction is being specified without an axis.
- In the top right, direction is being specified with a horizontal axis only.
- In the bottom center, direction is being specified with a vertical axis only.

Where objects are no closer than *min* and no further than *max*. If an axis is specified, the measurement is taken along the direction of the axis.

Note: Both *Min* and *Max* parameters have a default of (not set) which equates to 0 and `INT_MAX` respectively.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types



The figure above contains examples of the Distance constraint with single or bi-directional axis specified.

If more than two objects form part of a Distance constraint, the distance will be measured between the first object and any other member. The direction in which the distance is measured is unspecified, but wherever necessary should be specified with an ordered Alignment constraint.

The Distance constraint also uses a *reference* parameter that specifies whether a measurement is in respect to the closest edge orthogonal to the measurement direction (set as, *edge*), the object bBox center (*center*), or the origin points (*origin*).

Edge Pin Group Guide

Definition The Edge Pin Group Guide constraint defines the placement of pin members on an edge of the PR boundary.

Parameters In addition to the parameters *Name*, *Enabled*, and *Notes*, you can set the following parameters:

■ **Default Group (Constraint Group)**

The *Default Group* contains the following parameters: *Widths Minimum*, *Spacing*, *preserveOrder*, *spreadPins*, *edgeName*, *startOffset*, *endOffset*, and *pinGroupType*

- ❑ *Widths Minimum* defines the default minimum width for member pins in the pin group guide. Width is defined per layer in user units.
- ❑ *Spacing* defines the default minimum spacing for the pin group guide. Spacing is defined per layer in user units.
- ❑ *preserveOrder* defines whether the member pins are placed in the same order as the members.

Type: Boolean

Default: false

- ❑ *spreadPins* defines whether the member pins are evenly spaced.

Type: Boolean

Default: false

- ❑ *edgeName* defines the name of the PR Boundary edge. The *edgeName* choices are defined by the PR Boundary.
- ❑ *startOffset* defines the start of the boundary edge region in user units.
- ❑ *endOffset* defines the end of the boundary edge region in user units.

- ❑ *pinGroupType* defines whether the group is inclusive or exclusive. It specifies if other non-member pins are allowed in the regions specified by the PR Boundary edge start and end offsets.

Type: enum

Default: AllowForeignPins

Values: This parameter accepts one of the following values:

- *AllowForeignPins* means other pins are allowed in the whole region.
- *ExcludeOnAllLayers* means exclude other pins between members on all layers.
- *ExcludeOnPinLayers* means exclude other pins between members on pin layers.
- *ExcludeForeignPins* means exclude other pins from whole region on all layers.

■ Within Group (Constraint Group)

The *Within Group* contains the *Spacing* parameter. The Within Group parameters are used to enter process rules between pin members in the group.

- ❑ *Spacing* defines the minimum spacing between a pin and other member pins. Spacing is defined per layer in user units.

■ Group to Outside (Constraint Group)

The *Group to Outside* contains the *Keep Out Spacing* parameter. The Group to Outside parameters are used to enter process rules between pin members and other pins in the design.

Keep Out Spacing defines the minimum spacing between the pin group guide members and other pins. Spacing is defined per layer in user units.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Members

The *Edge Pin Group Guide* members are one or more pins, plus the PR boundary. The PR boundary edge and offsets define the region in which the pin members can be placed.

Supporting Applications

■ Virtuoso Floorplanner

- Pin Group Guide commands to Create, Edit, and Compare

For detailed information, refer to the *Creating Pin Group Guides* chapter in the *Virtuoso Floorplanner User Guide*.

- Pin Planning and Pin Optimization

For detailed information, refer to the following sections of the *Pin Placement* chapter in the *Virtuoso Floorplanner User Guide*:

- *Pin Planner* section
- *Pin Optimization* section

■ Innovus System

Note: This constraint type is interoperable with Innovus and validated by PVS-CV.

Fixed

Definition Sets the placement status on instances and pins, and the routing status on paths, pathSegs, and vias.

You can create a Fixed constraint on a Modgen or a figGroup, which in turn creates the necessary Fixed constraint status on the individual member objects.



The Fixed constraint does not support parameter notes and history (timestamp) tracking (see [Constraint History](#)). Any fixed constraints that are out-of-context will be purged when the cache is closed.

Parameters *Name* information can be set. Constraint *Owner* (Lib/Cell/View) information is also provided

Members One or more instances, pins, nets or oaPinFigs(dbPinFig).

Supporting Applications

- Virtuoso Analog Auto Placer
- Constraint Aware Editing
- Virtuoso Custom Digital Placer
- Virtuoso Floorplanner
- Virtuoso Space-based Router

Note: The fixed constraint cannot be stored as a template as it is an Open Access object attribute and does not have independent OA storage. For more information, see [Constraint Template Storage](#) in the Virtuoso Unified Custom Constraints Configuration Guide.

Adding and Deleting Nets from a Fixed Constraint

When a net is **added** to a *fixed* constraint, all existing shapes on that net will be marked as fixed. However, any new shapes added to that net, after a *fixed* constraint has already been applied, will not be deemed to be fixed unless the *fixed* constraint is again re-applied.

A net will appear in a *fixed* constraint if any of its shapes are marked as fixed. When a net is **deleted** from a *fixed* constraint, all shapes will be marked as *normal*.

Note: You can use the *Property Editor* assistant or constraint haloing to see which shapes are fixed (where only fixed shapes will be haloed).

Guard Ring

Definition

The Guard Ring constraint defines rails, rings or guard rings for a modgen or boundary. Its members are either a modgen or a boundary and any number of nets for each of the rings or stripes of the guard ring.

You can create a guard ring constraint by selecting either a modgen or a cluster constraint in the *Constraint Manager*, along with the relevant net.

If the parameter *mppName* is specified a multi-part path, as defined in the technology file, will be used for the guard ring. Otherwise *layerNumber* must be specified to define the layer for the guard ring. In both cases the parameter *width* defines the width of the multi-part path (MPP) or the layer.

The *offset* parameter defines the distance to be kept between modgen or boundary and guard ring.

The *spacing* parameter on members specifies the distance between members of the guard ring. This follows the order of the members. For example, the first member is a modgen or boundary, and the second, third, and so on, members are nets. The parameter *spacing* on the second member, which is the first net, refers to the spacing between modgen or boundary (first member) and the first net (second member). All subsequent nets get a parameter spacing that is relative to the previous net.

Offset is a parameter used in combination with rails to determine the rail offset from the boundary in case the rail does not abut the design boundary. When an *offset* is specified, it takes precedence over the *spacing* parameter.

If the parameter *chamfer* has a value greater than zero, 45 degree angles will be used at each corner of the guard ring. The value of *chamfer* then gives the size of the 45 degree corner.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Parameters

In addition to the parameters *Status*, *History*, *Name*, *Enabled*, and *Notes*, you can set the following parameters:

■ **shape**

Type: enum

Default: rectangular

Valid Values: “rectangular”, “rectilinear”

On Constraint: N

On Member: Y

Display: Y

User Units: N

■ **mppName**

Type: cyclic

Valid Values: drop-down list box that lists all the MPPs in the technology file.

Note: A number of other *Guard Ring* parameter fields, for example *offset*, *layer* and *chamfer*, will not have an impact when the guard ring is a MPP that has been created using the Modgen user-interface.

On Constraint: N

On Member: Y

Display: Y

User Units: N

■ **spacing**

Type: int

Default: 0

Valid Values: separation of this ring from contents

On Constraint: N

On Member: Y

Display: Y

User Units: Y

■ **leftSpacing**

Type: int

Default: 0

Valid Values: left separation of this ring from contents.

On Constraint: N

On Member: Y

Display: Y

User Units: Y

■ **layerNumber**

Type: string

Valid Values: layer number when not MPP.

On Constraint: N

On Member: Y

Display: Y

User Units: Y

■ **width**

Type: int

Default: 0

Valid Values: layer width when not MPP.

On Constraint: N

On Member: Y

Display: Y

User Units: Y

■ **rightSpacing**

Type: int

Default: 0

Valid Values: right separation of this ring from contents.

On Constraint: N

On Member: Y

Display: Y

User Units: Y

■ **bottomSpacing**

Type: int

Default: 0

Valid Values: bottom separation of this ring from contents.

On Constraint: N

On Member: Y

Display: Y

User Units: Y

■ **topSpacing**

Type: int

Default: 0

Valid Values: top separation of this ring from contents.

On Constraint: N

On Member: Y

Display: Y

User Units: Y

■ **offsets**

Type: intRange

Default: (0 0)

Valid Values: VCP offsets

On Constraint: N

On Member: Y

Display: Y

User Units: Y

■ **chamfer**

Type: int

Default: 0

Valid Values: positives

On Constraint: N

On Member: Y

Display: Y

User Units: Y

Members

A single modgen or cluster to put the guard ring around any number of nets, one for each guard ring. The same net may appear multiple times.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Supporting Applications

- Virtuoso Analog Auto Placer
- Virtuoso Layout Suite XL (Constraint Aware Editing)
- Modgen Editor (shape, mppName, spacing parameters only)

Note: The only parameters currently supported are *shape* (both values), *mppName* and the various spacing parameters. Note also that the Analog Placer and Constraint Aware Editing will support *layerNumber* and *width*, rather than *mppName*.

High Precision C Extraction

Definition The *High Precision C Extraction* constraint is used to specify the nets that should be extracted with higher accuracy and define a convergence goal (optional) to be used by the extraction engine.

Parameters In addition to the parameter given below, you can set the *Name* parameter.

■ **Convergence Target**

Type: float

Default: 1

Valid Values: any numerical value greater than 0

On Constraint: Y

On Member: N

Display: Y

User Units: N

Members Minimum of one net.

Supporting Applications ■ Electrically Aware Design (EAD) Flow

High Precision R Extraction

Definition The *High Precision R Extraction* constraint is used to specify the nets that should be extracted with higher accuracy and define the mesh density (optional) required for the extraction engine.

Parameters In addition to the parameter given below, you can set the *Name* parameter.

■ **Mesh Density**

Type: float

Default: 1

Valid Values: any numerical value greater than 0

On Constraint: Y

On Member: N

Display: Y

User Units: N

Members Minimum of one net.

Supporting Applications ■ Electrically Aware Design (EAD) Flow

Locked

Definition Sets the placement status on instances and pins, and the routing status on paths, pathSegs, and vias.

You can create a Locked constraint on a Modgen or a figGroup, which in turn creates the necessary Locked constraint status on the individual member objects.



The Locked constraint does not support parameter notes and history (timestamp) tracking (see [Constraint History](#)). Any locked constraints that are out-of-context will be purged when the cache is closed.

Parameters *Name* information can be set. Constraint *Owner* (Lib/Cell/View) information is also provided

Members One or more instances, pins, nets, or oaPinFigs(dbPinFig).

Supporting Applications

- Virtuoso Space-based Router
- Virtuoso Analog Auto Placer
- Virtuoso Custom Digital Placer
- Virtuoso Layout Suite XL (Constraint Aware Editing)
- Virtuoso Layout Suite XL (Create Wire Editing)
- Virtuoso Floorplanner

Note: The locked constraint cannot be stored as a template as it is an Open Access object attribute and does not have independent OA storage. For more information, see [Constraint Template Storage](#) in the *Virtuoso Unified Custom Constraints Configuration Guide*.

Adding and Deleting Nets from a Locked Constraint

When a net is **added** to a *locked* constraint, all existing shapes on that net will be marked as locked. However, any new shapes added to that net, after a *locked* constraint has already been applied, will not be deemed to be locked unless the *locked* constraint is again re-applied.

A net will appear in a *locked* constraint if any of its shapes are marked as locked. When a net is **deleted** from a *locked* constraint, all shapes will be marked as *normal*.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

You can use the *Property Editor* assistant or constraint haloing to see which shapes are locked (where only lock shapes will be haloed).

Matched Capacitance

Definition The *Matched Capacitance* constraint is used to define the absolute or relative difference permissible between total or coupled capacitances for the selected nets.

Parameters In addition to the parameters *Name*, *Enabled*, and *Notes*, you can set the following parameters:

■ **Diff. Type**

Type: enum

Default: absolute

Valid Values: absolute, percent

On Constraint: Y

On Member: N

Display: Y

User Units: N

■ **Difference**

Type: float

Default: 0.0

Valid Values: for *absolute*, the difference should be greater than or equal to 0; for *percent*, the difference should be greater than or equal to 0, but less than or equal to 100

On Constraint: Y

On Member: N

Display: Y

User Units: N

Members Minimum two unique nets. The same net can be used in any number of *Matched Capacitance* constraints; however, the same combination of nets cannot be used more than once.

Supporting Applications

- Electrically Aware Design (EAD) Flow

Matched Instance Color

Definition	Checks whether the instances have matching color.
Parameters	In addition to the parameters <i>Name</i>, <i>Enabled</i>, and <i>Notes</i>, you can set the following parameters: ■ Align Mask Color <i>Type: boolean</i> <i>Default: true</i> <i>On Constraint: (Y/N)</i> <i>On Member: (Y/N)</i> <i>Display: (Y/N)</i> <i>User Units: (Y/N)</i>
Members	The <code>matchedInstanceColor</code> constraint type cannot share members with the same constraint type.
Supporting Applications	None.

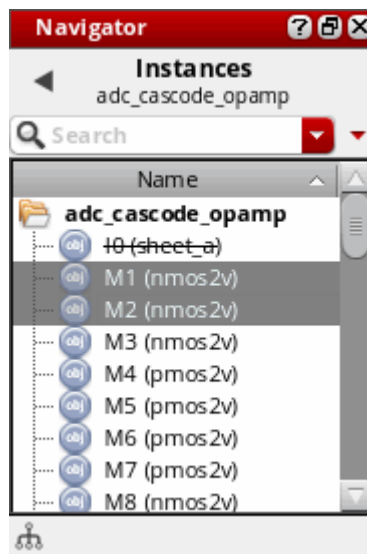
Matched LDE Parameters

Definition

The Matched Parameters constraint performs a matching analysis of the drive current across the schematic and the layout views.

To create the Matched LDE Parameters, you need to perform the following steps:

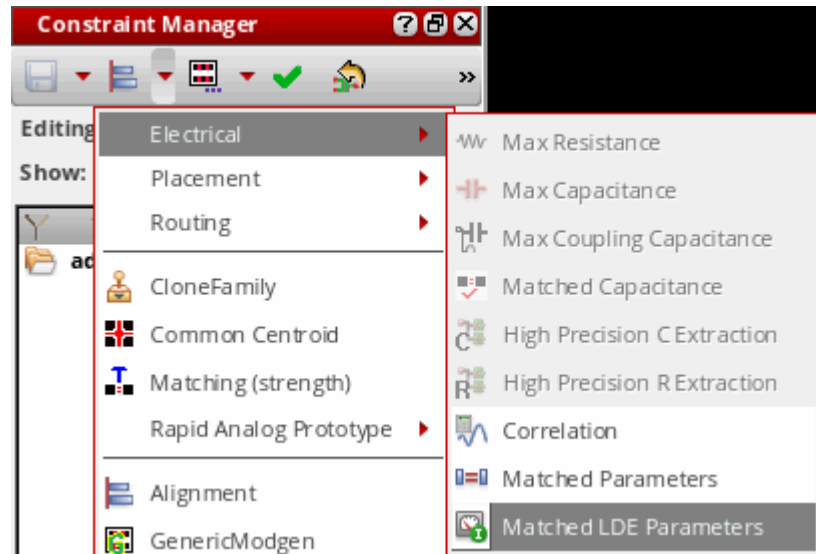
1. Select at least two instances from the Navigator assistant or the design area.



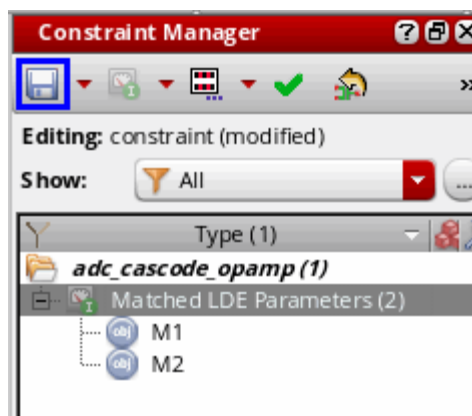
Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

2. In the Constraint Manager assistant toolbar, select *Electrical-Matched LDE Parameters* option.



3. Select the constraint created in the Constraint Manager to display the *Constraint Parameter Editor* pane.
4. Now, to save the constraint, click the *Save* icon in the Constraint Manager toolbar, as shown below.



Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Parameters

In addition to the parameters *Name*, *Enabled*, and *Notes*, you can set the following parameters:

■ Tolerance

Type: string

On Constraint: Y

On Member: N

Display: Y

User Units: N

■ Mode

Type: enum

On Constraint: Y

On Member: N

Display: Y

User Units: N

Note: The MATCH mode compares across layout devices. For example, if a matching constraint has a threshold of 5%, then any two devices in layout with a saturation current difference of greater than 5% are flagged. The schematic devices are ignored.

In the GROUP mode, each layout device is compared with its schematic version and anything greater than the specified threshold is flagged. Similar to the LDE analysis command carried out using the main LDE GUI, except this comparison is only applied for the selected devices. Use GROUP mode when a general schematic matching of 5% is obtained using the LDE GUI, as this comparison could do a 1% (tighter) check on a sub-set of devices in the layout.

■ **Ratio**

Type: float

On Constraint: N

On Member: Y

Display: Y

User Units: N

- *Range:* 0.0 to `maxFloat`. Here, `maxFloat` is the maximum float value displayed when you move the pointer over the parameter in the Constraint Manager assistant.

■ **Comparison**

Type: enum

On Constraint: Y

On Member: N

Display: Y

User Units: N

Note: The *Comparison* parameter is a dynamic parameter that is hidden by default. To make it visible in the parameter list, perform the following steps in the `config.xml` file:

- a. Search the `Param` tag associated to the Matched LDE Parameter constraint.
- b. Reset the `visibleInEditor` attribute to `true` in the `GUIName` tag within this `Param` tag.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

■ Method

Type: enum

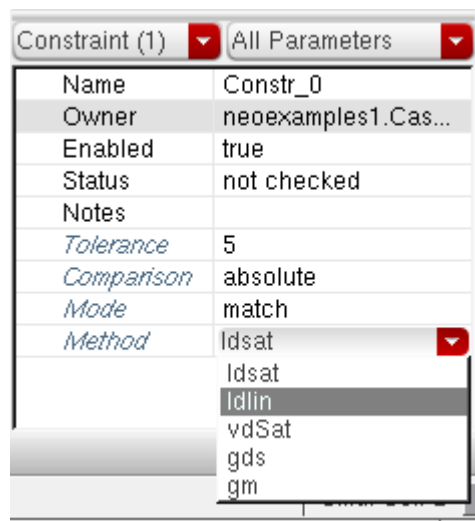
On Constraint: Y

On Member: N

Display: Y

User Units: N

The *Method* parameter is a dynamic parameter that can be added using the `ciRegisterDynamicParamDef` function and appears in the *Constraint Parameter Editor* pane as shown in the figure below. It can be used to fetch the method list from the list previously defined in `libInit.il`.



The *Method* parameter works as the differentiating factor between two Matched LDE constraints with same set of members. However, this happens only when you enable this parameter.

Members

Instances of a Device.

Supporting Applications

- Virtuoso Layout Suite XL
- Virtuoso Schematic Editor XL

Matched Length

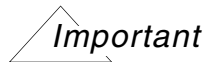
Definition

A Matched Length constraint defines a specialty routing relationship between two or more nets. The nets should have the equivalent lengths.

It is used to choose the style that the router should use to match length. Each constraint parameter corresponds to the existence of the same named constraint in the Matched Length default constraint group.

For more information, see [The Process Rule Editor](#).

Parameters



See also [Matched Length Parameters and the Process Rule Editor](#).

In addition to the parameters *Status*, *History*, *Name*, *Enabled*, and *Notes*, you can set the following parameters:

■ Default Group (Constraint Group)

The *Default Group* contains the following parameters: *Top Layer*, *Bottom Layer*, *Purposes*, *Vias Allowed*, *Widths Minimum*, *Cuts Minimum*, *Spacing*.

The *Default Group (Constraint Group)* is used to enter process rules containing parameters that are concerned with how nets should be matched.

These constraint group parameters include information on the layers to use for Matched Length. Parameters correspond to a valid layers constraint in the PRE. The layers constraints should always contain a contiguous set of layers to use for this net (/net group). The router should route using only these layers.

Additionally, the spacing (for each layer) represents the min spacing default constraint group for Matched Length. Therefore, the *Default Group* also includes the following parameters: *lengthPatternAccordion*, *lengthPatternDangle*, *lengthPatternRWAccordion*, *lengthPatternTrombone*, *lengthPatternEndRun*, *msMatchPerLayer*, *routeMinLength*, and *routeMaxLength*.

In addition to the parameters listed above, *Default Group* has the *Wire Types* parameter that can be used to define a set of wire types for each layer. For more information, see [Assigning a Wire Type to a Net](#).

For more information on the Default Group parameters, see [The Process Rule Editor](#).

■ Within Group (Constraint Group)

The *Within Group* contains the following parameters: *Spacing*, *matchTolerance*, and *Tolerance*.

For more information on Within Group, see [The Process Rule Editor](#).

■ Group to Outside Group (Constraint Group)

The *Group to Outside Group* contains the following constraints: *Spacing* and *hierarchicalScope*.

Here, the Group to Outside Group constraint group parameters are used to enter process rules between nets in a Matched Length constraint, and those nets that are not part of that constraint.

For more information on Group to Outside Group, see [The Process Rule Editor](#).

Members

Two or more nets.

Note: A given net may belong to only 1 specialty routing constraint (for example, [Symmetry](#), [Diff Pair](#), or [Matched Length](#)).

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Supporting Applications

- Virtuoso Space-based Router
- Innvus System

Note: VLO/VLM support is currently limited to “matching the length”. It does not, for example, look at Default or Group to Outside Group parameters.

Matched Length Parameters and the Process Rule Editor

The following Matched Length parameters allow to edit the process rules in [The Process Rule Editor](#) (PRE). That is, changing the value of these parameters in the Constraint Manager assistant make equivalent changes in the PRE and conversely. Each parameter tooltip also provides information on where, in the constraint lookup, a value was found.

Matched Length Parameter	PRE Equivalent Parameter Name	Notes
length Pattern Accordion, length Pattern RW Accordion, length Pattern Trombone, length Pattern End Run	constraints by the same name in matched length default constraint group.	Specifies the <i>type</i> of length pattern that routers should use to match length. The router should only attempt to use the patterns that have been checked for use (or all if no patterns have been checked).
Note: All of the above parameters are boolean.		
match per layer (boolean), match tolerance (int), route min length (int), route max length (int)	constraints by the same name in the matched length default constraint group.	

Matched Orientation

Definition	The Matched Orientation constraint implies that the <u>Orientation</u> or the direction is to be matched (that is, identical) for the member instances.
Parameters	<i>Name</i> , <i>Enabled</i> , <i>Notes</i> , and <i>Match</i> information can be set.
Members	Two or more instances, modgen constraints or figGroups, or regular figGroups. Member order is not currently important, given that "R0" is the only valid orientation value.
Supporting Applications	■ Virtuoso Analog Auto Placer ■ Virtuoso Layout Suite XL (Constraint Aware Editing)

Using the Match Parameter of Matched Orientation Constraint in Layout

The *Match* parameter of the Matched Orientation constraint helps you to evaluate the selected member instances based on two match criteria, *Orientation* and *Direction*.

Important

The Matched Orientation constraint can be seen in both schematic and layout. However, only in layout, you see a *Status* parameter that displays whether the matching *passed* or *failed* on the basis of the specified match criteria.

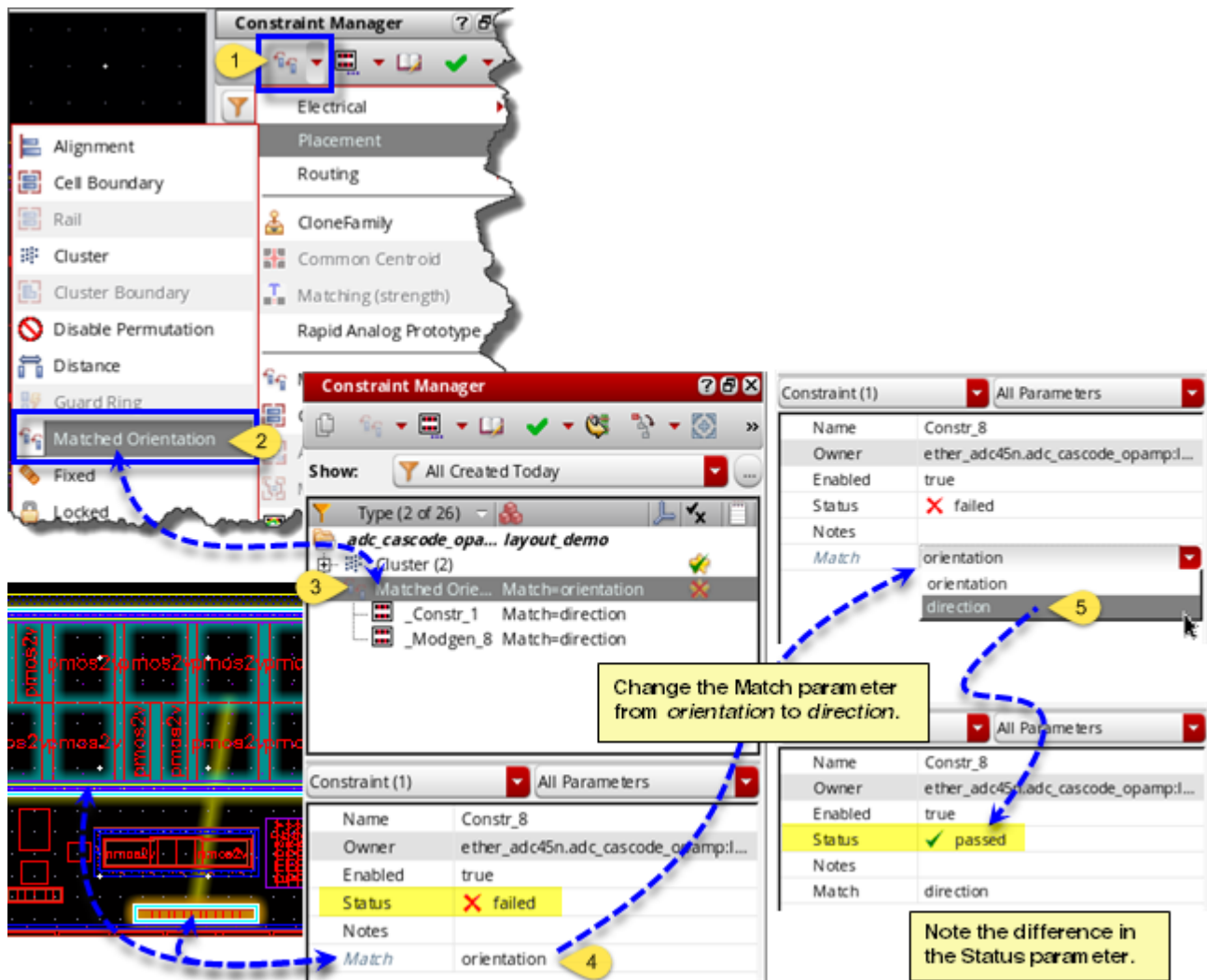
The following steps show how you can use the *Match* parameter in layout.

1. Select the member instances that need to be evaluated.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

2. Choose *Constraint Creation — Placement — Matched Orientation*. Refer to the figure below.

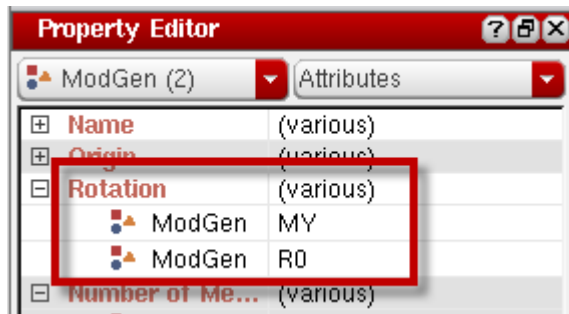


The generated Matched Orientation constraint is displayed in the Constraint Browser, as shown in the figure above. By default, the *Match* parameter is set to *orientation*. If the orientation of the selected member instances matches, the *Status* parameter displays *passed*. Otherwise, the *Status* is displayed as *failed*. In the latter case, you can initiate re-evaluation by resetting the *Match* parameter to *direction* from the list box, as shown above. The selected member instances then get evaluated on the basis of direction and the *Status* changes to *passed* if their orientation matches.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

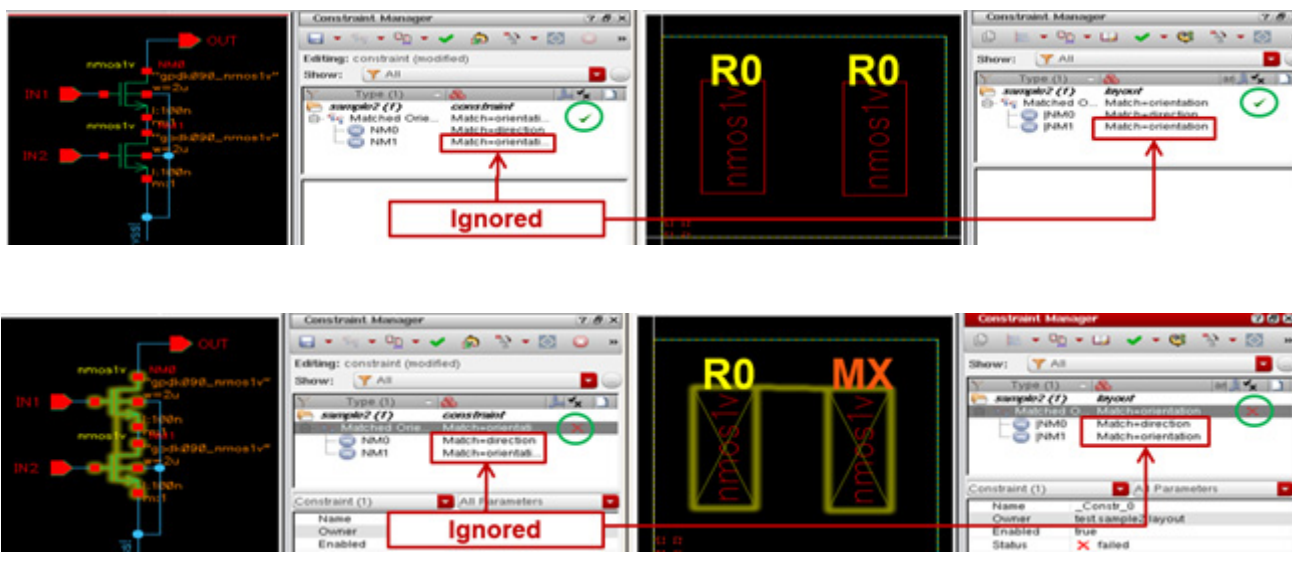
Note: In the Property Editor, you can use the *Rotation* attribute to check and update the orientation or direction values of the selected member instances. For more information, see the *Property Editor* in the *Virtuoso Schematic Editor User Guide*



The members of a Matched Orientation constraint can also have the parameter, *Match*, that too accepts one of the following values: *orientation* or *direction*.

The table below lists the expected behavior when the *Match* parameter is set for a Matched Orientation constraint and is set or not set for its member(s):

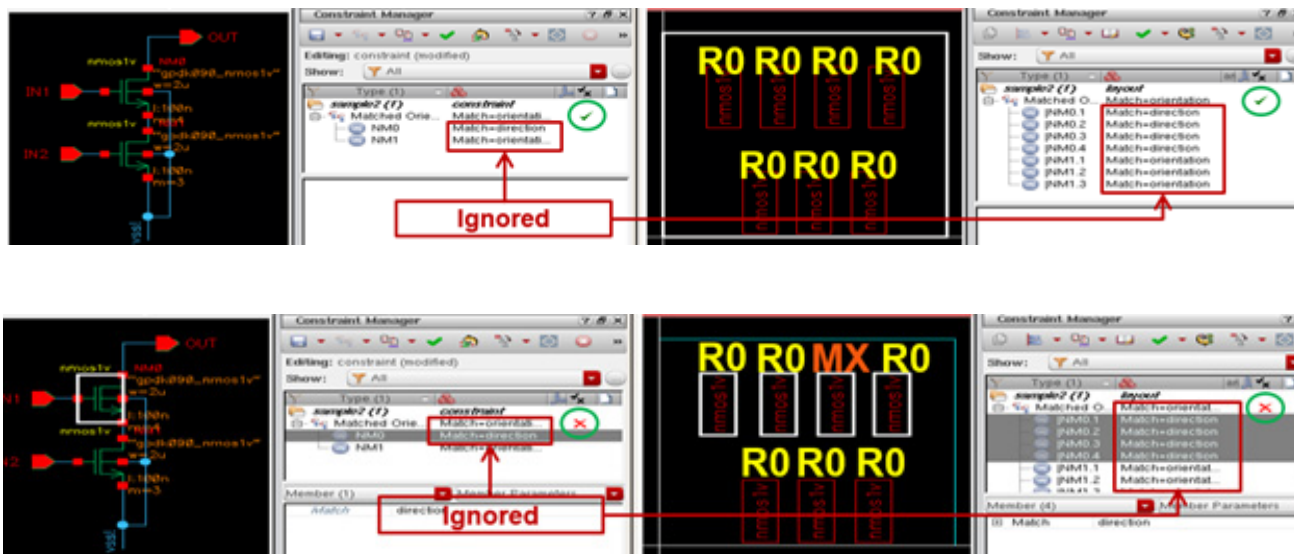
Constraint's Match Parameter	Device's mFactor or iteration	Member's Match Parameter	Behavior
<i>orientation</i>	=1	not set	All devices share the same orientation, whatever it is, as shown in the examples below.



Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

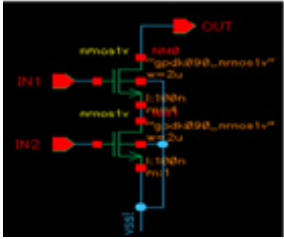
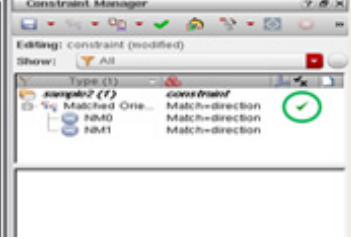
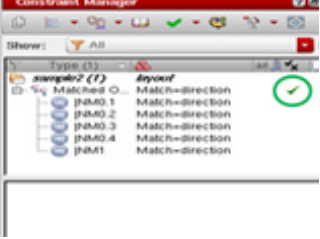
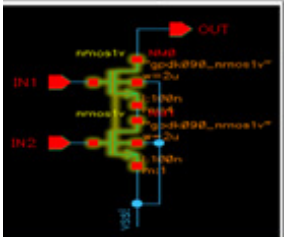
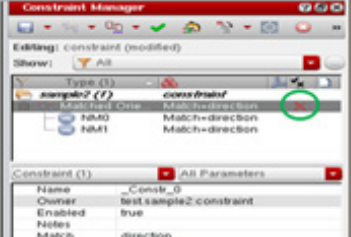
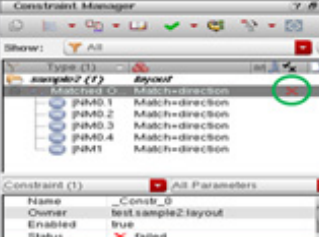
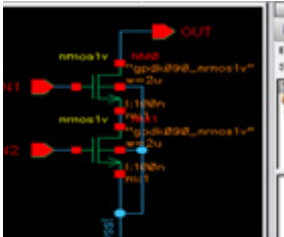
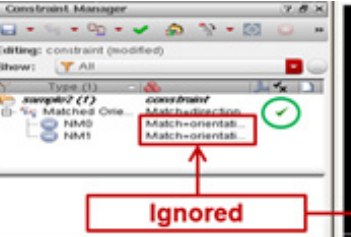
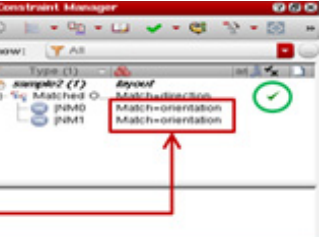
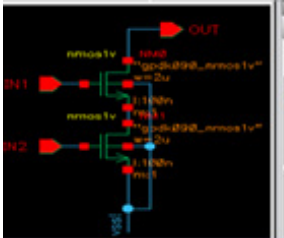
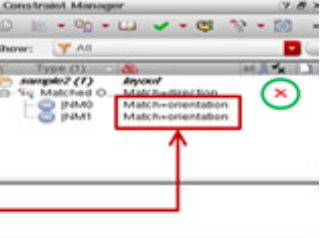
Constraint's Match Parameter	Device's mFactor or iteration	Member's Match Parameter	Behavior
<i>orientation</i>	>1	not set	All devices share the same orientation, whatever it is, as shown in the examples below.



<i>direction</i>	≥ 1	<i>direction</i>	All devices can have multiple orientations as long as they share the same direction, as shown in the examples below.

Virtuoso Unified Custom Constraints User Guide

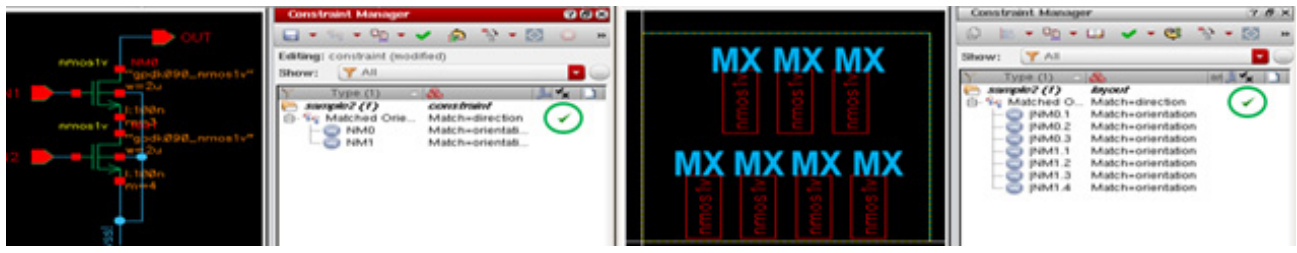
Default Constraint Types

Constraint's Match Parameter	Device's mFactor or iteration	Member's Match Parameter	Behavior
			
			
<i>direction</i>	=1	not set	Constraint's members take the same direction; while the members' <i>Match</i> parameter has no meaning, as shown in the examples below.
			
			

Virtuoso Unified Custom Constraints User Guide

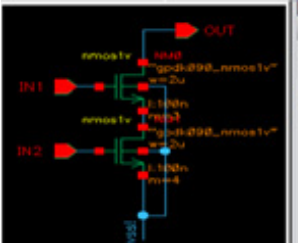
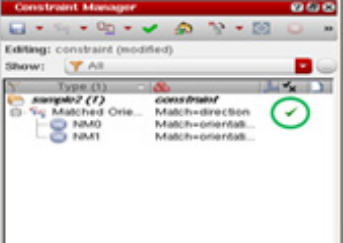
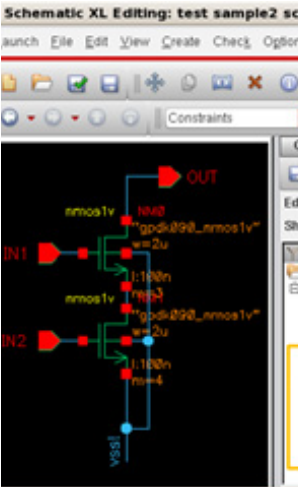
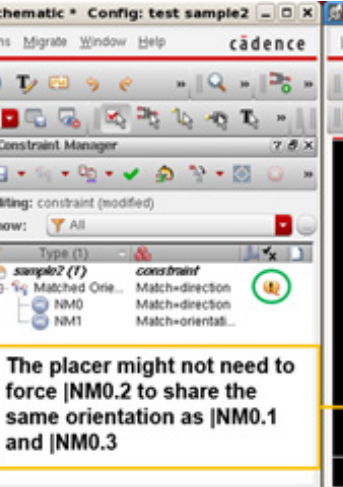
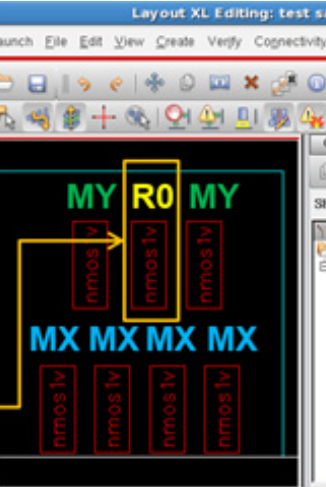
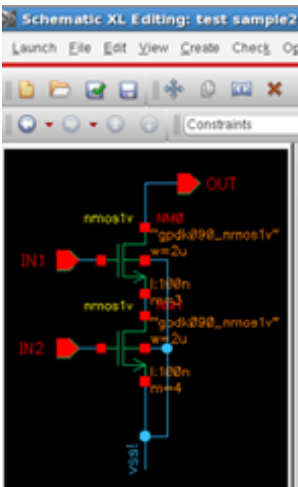
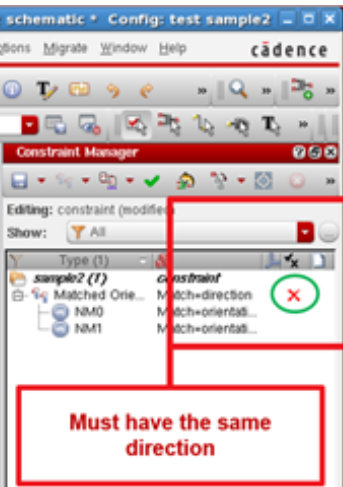
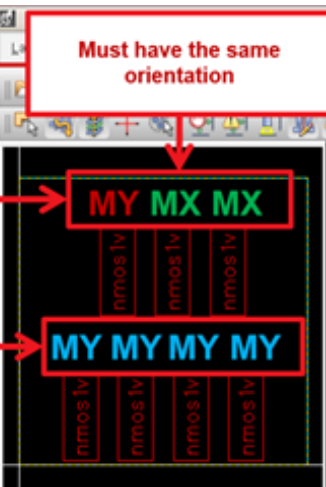
Default Constraint Types

Constraint's Match Parameter	Device's mFactor or iteration	Member's Match Parameter	Behavior
<i>direction</i>	>1	<i>orientation</i>	Constraint's members take the same direction; while the members' mFactored devices share the same orientation, as shown in the examples below.



Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Constraint's Match Parameter	Device's mFactor or iteration	Member's Match Parameter	Behavior
			
			
			

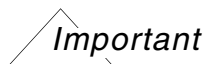
Matched Parameters

Definition The Matched Parameters constraint defines which parameters must match with respect to its members.

By default, all parameters must match, but a subset of parameters to be matched can also be specified using the matchSubset constraint parameter.

Note: When specifying a subset of parameters to match, names must be separated by one or more space and/or comma characters. No quotes or double quotes should be used.

If a ratio parameter is set on the first parameter, a warning will be issued by the consistency checker. The constraint will still, however, be accepted and the ratio value ignored for the first member.



For information on how you can check for any mismatched parameters, using the Setup Schematic Rules Check form, see [Constraints Checks](#) in the *Virtuoso Schematic Editor User Guide*.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Parameters

In addition to the parameters *Name*, *Enabled*, and *Notes*, you can set the following parameters:

■ **Match Subset** (see also [Setting Match Subset Parameter Values](#))

Type: string1DTbl

On Constraint: Y

On Member: N

Display: Y

User Units: N

The Match Subset parameter is used to specify the list of parameter names that must have the same respective value (or ratio value) for all members of the constraint. If a parameter does not exist for one of its members its value is `nil`. A non-existing parameter is considered as a match only if the parameter name is not defined for any of the members. All values for the given parameter names must match - or ratio match - respectively for the constraint to *pass*. If the Match Subset parameter does not specify any parameter name (empty), the values of all existing parameter names of the constraint members must match - or ratio match - respectively.

■ **ratio** (see also [Ratio Parameter](#))

Type: float

Default: 1.0

Valid Values: float

On Constraint: Y

On Member: Y

Display: Y

User Units: N

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

The matching of parameters may not always be defined with a 1:1 ratio. For example, the ratio of the number of fingers for each device member of a current mirror versus the driver member (reference) can be the same ratio of the respective current that they each drive.

The ratio parameter is defined as:

$$\text{Ratio} = (\text{parameter value of the member}) / (\text{parameter value of the reference})$$

Members

Two or more instances.

Note: A reference member will always have a ratio parameter setting of 1 (see [Ratio Parameter](#)).

Supporting Applications

- Virtuoso Analog Auto Placer

Note: Does not support the ratio parameter.

- Virtuoso Layout Suite XL (Constraint Aware Editing)

Setting Match Subset Parameter Values

The *Match Subset* parameter field lists all the parameter names whose corresponding values must match - or ratio match - for all object members.

By default, the list of parameter names to be matched is set by the following default procedure:

```
procedure( ciDefaultParamToMatchFilter(inst simulator)
  prog((param2DList)
    if( ciIsDevice(inst "fet") then
      param2DList= ciGetMatchParam2DList(inst "fetParamListForMatching")
    )
    if( ciIsDevice(inst "resistor") then
      param2DList = ciGetMatchParam2DList(inst "resistorParamListForMatching")
    )
    if( ciIsDevice(inst "capacitor") then
      param2DList = ciGetMatchParam2DList(inst
"capacitorParamListForMatching")
    )
    return (param2DList)
  )
)
```

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

with the list of parameter names per device type defined by:

```
ciMapParam("fetParamListForMatching" ' ("model" "w" "l" "m"))
ciMapParam("resistorParamListForMatching" ' ("model" "w" "l" "r" "m"))
ciMapParam("capacitorParamListForMatching" ' ("model" "w" "l" "c" "m"))
```

You can override the above default filter with a different [ciDefaultParamToMatchFilter](#) function or create a new filter associated with a new function and register it with:

```
ciRegisterCustomDeviceFilter(name func)
```

For example:

```
ciRegisterCustomDeviceFilter("myDefault" `CstMyFilter)
```

In the *Constraint Manager* editor, you can also change these values manually by editing the *Match Subset* parameter field directly. Alternatively, you can edit this list using the Edit Match Subset form which tabulates different sets of parameter names to be matched.

To edit *Match Subset* parameter values in the Edit Match Subset form:

1. Select the ... (*Click to expand editor*) option adjacent to the *Match Subset* parameter.

This will display the Edit Match Subset form.

2. In the Edit Match Subset form, select/deselect the parameters that you want to have set by checking/unchecking the available parameters.

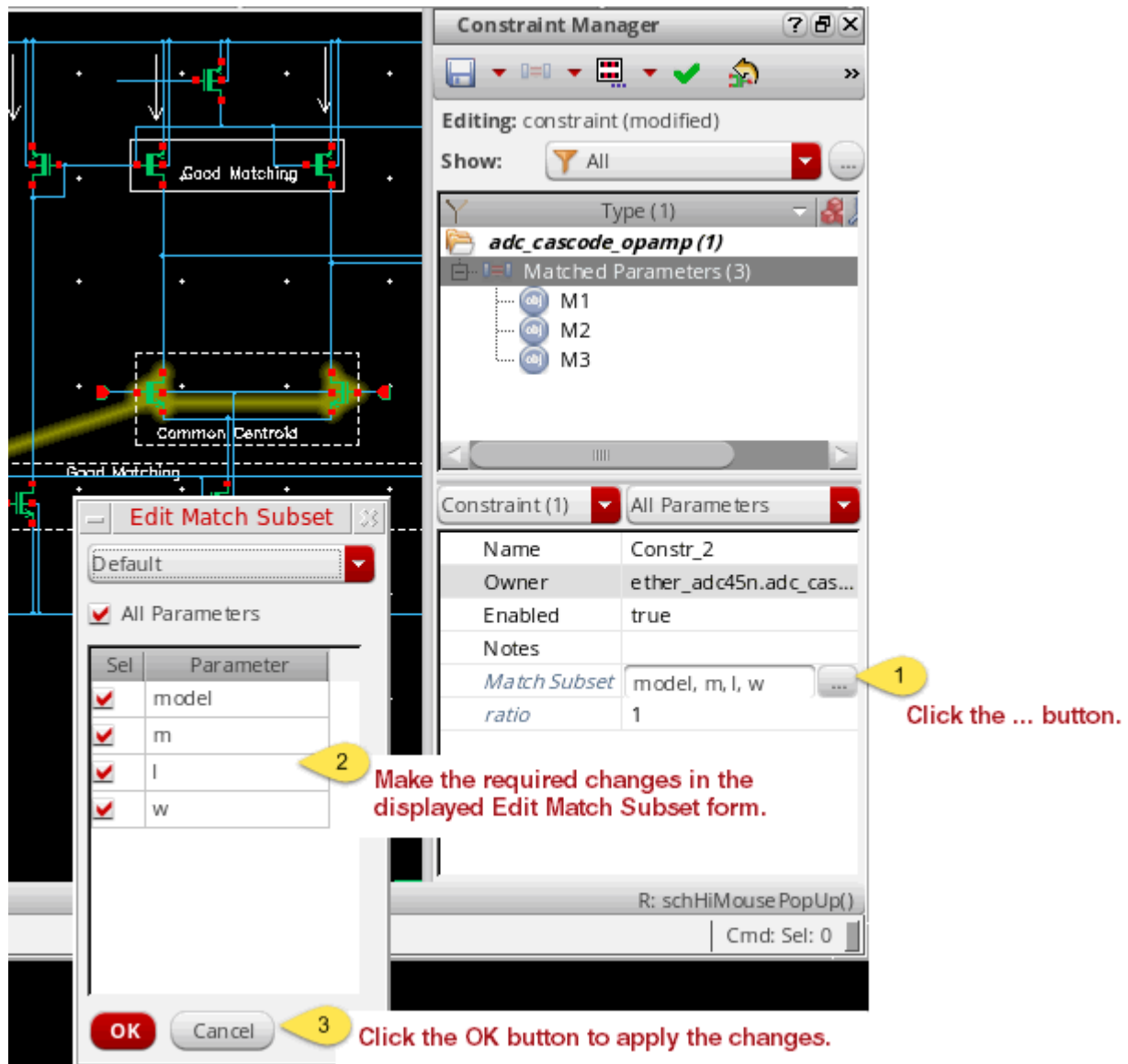
Note: You can narrow the parameter options available by selecting a filter (*CDF Parameters*, *CDF Editable*, or *Default*) from the *Default* drop-down list box, or also quickly select or deselect all of the available parameters by checking the *All Parameters* option.

3. Once your selection is complete, click *OK*.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Your new selection will now be displayed in the *Match Subset* parameter field.



Ratio Parameter

Ratio can be a constraint parameter and its value applies to all members except the first one. Alternatively, it can be a member parameter.

Its value is set to 1 by default.

- The first member is considered as the reference. The value of *Ratio* for the first member is ignored or always considered as 1.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

- For the other members, the value of *Ratio* is used as a multiplication factor to check whether the value of the parameters given as the *matchSubset* parameter effectively match or not.

For example, if *NumberOfFingers* is set as *matchSubset* and the *Ratio* of the second member is set to 4, the value of *NumberOfFingers* for that member must be 4 times the value of *NumberOfFingers* for the first, or *reference*, member.

Also, if you changed the *Ratio* value in the parameter editing table to, for example, 0.7, this will be reflected in the *Parameters* column of the Constraint browser for each constraint member.

Constraint Manager

Editing: constraint (modified)

Show: All

Type (1)

adc_cascode_opamp (1) constraint

Matched Parameters (3)

- M1 ratio=0.7
- M2 ratio=0.7
- M3 ratio=0.7

Constraint (1) All Parameters

Name	Constr_2
Owner	ether_adc45n.adc_cascode_opamp:constraint
Enabled	true
Notes	
Match Subset	model, m, l
ratio	0.7

Matched Parameter constraint members ratio parameter set to 0.7.

Ratio parameter value set to 0.7.

Max Capacitance

Definition The *Max Capacitance* constraint is used to define the permissible maximum capacitance of the following types for the selected net:

- Total capacitance
- Coupled capacitance
- Ground capacitance

Parameters In addition to the parameters *Name*, *Enabled*, and *Notes*, you can set the following parameters:

- **Total Capacitance**

Type: float

Default: 5f (5e - 15)

Valid Values: any numerical value greater than 0

On Constraint: Y

On Member: N

Display: Y

User Units: N

- **Coupled Capacitance**

Type: float

Default: 0.0

Valid Values: any numerical value greater than 0

On Constraint: Y

On Member: N

Display: Y

User Units: N

■ **Ground Capacitance**

Type: float

Default: 0.0

Valid Values: any numerical value greater than 0

On Constraint: Y

On Member: N

Display: Y

User Units: N

Members

Maximum one net. If you select more than one net, the maximum capacitance will be defined for each net separately.

**Supporting
Applications**

Electrically Aware Design (EAD) Flow

Max Coupling Capacitance

Definition	The <i>Max Coupling Capacitance</i> constraint is used to define the maximum coupled capacitances between one of the following: two nets, two net classes, or a combination of a net and a net class.
Parameters	In addition to the parameters <i>Name</i>, <i>Enabled</i>, and <i>Notes</i>, you can set the following parameter: ■ Coupled Capacitance <i>Type</i>: float <i>Default</i>: 0.0 <i>Valid Values</i>: any numerical value greater than 0 <i>On Constraint</i>: Y <i>On Member</i>: N <i>Display</i>: Y <i>User Units</i>: N
Members	Two unique nets or net classes. A combination of a net and a net class is also accepted. Note: The same net or net class can be used in any number of <i>Max Coupling Capacitance</i> constraints; however, the same combination of nets cannot be used more than once directly or indirectly using net classes.
Supporting Applications	Electrically Aware Design (EAD) Flow

Max Resistance

Definition

The *Max Resistance* constraint is used to define the maximum resistance permissible between either two terminals or instance terminals or, between a terminal or instance terminal and all the terminals or instance terminals connected to it.

Note: This constraint can only be created on the schematic and transferred to layout because VLS XL does not allow selection of instance terminals.

Parameters

In addition to the parameters *Name*, *Enabled*, and *Notes*, you can set the following parameter:

■ Max Resistance

Type: float

Default: 0.5

Valid Values: any numerical value greater than 0

On Constraint: Y

On Member: N

Display: Y

User Units: N

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Members

There are two options available depending on the terminals selected:

- One terminal. The constraint is paired between the selected terminal or instance terminal and each terminal or instance terminal connected to it. The selected terminal works like a reference.
- Two unique terminals or instance terminals that are attached to the same net. In this case, a terminal or instance terminal can be used in any number of *Max Resistance* constraints; however, a pair of the same members cannot be used more than once.

Important Points to Note:

- When you create a *Max Resistance* constraint with one member terminal or instance terminal, it should not be included in any other two member constraint.
- You can create more than one single-member *Max Resistance* constraint using different terminals or instance terminals on a common net only if they have the same maximum resistance value. Multiple single-member constraints on a net cannot have different maximum resistance values.

Supporting Applications

Electrically Aware Design (EAD) Flow

Max Voltage Drop

Definition The *Max Voltage Drop* constraint is used to specify the maximum voltage drop between a reference instance terminal/pin and one or more other instance terminals/pins.

Parameters In addition to the parameters *Name*, *Enabled*, and *Notes*, you can set the following parameters:

■ **Voltage**

Sets the default voltage drop for all member instance terminals/pins with respect to reference instance terminal/pin.

Type: float

Default: 500m

Valid Values: any float value in decimal, engineering, scientific notation, or ratio format.

On Constraint: Y (Voltage specified on the constraint is the default for all members.)

On Member: Y (Voltage specified on a member overrides the default for that member.)

Display: Y

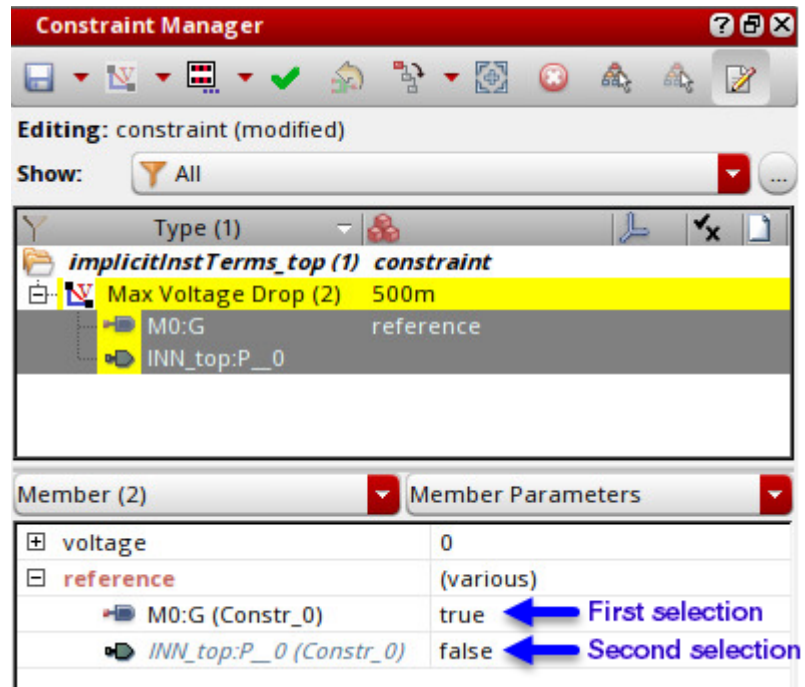
User Units: Y

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

■ reference

Resets the member instance terminal/pin as a reference instance terminal/pin if set to `true`.



Note: By default, the *reference* parameter of the first selected instance terminal/pin is set to `true` to identify it as the reference and the other member instance terminals/pins have this parameter set to `false`.

Type: Boolean

Default: `true` for first selected instance terminal/pin and `false` for the other instance terminals/pins

Valid Values: `true` or `false`

On Constraint: N

On Member: Y

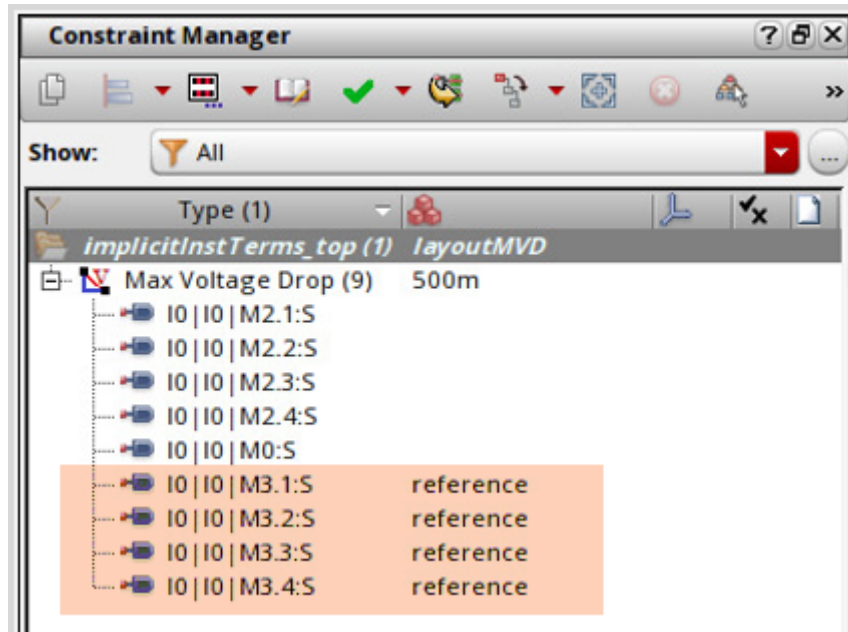
Display: Y

User Units: N

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

In layout, due to mfactors or fingers, there can be multiple reference instance terminals that are associated with a single schematic instance terminals.



Members

One or more unique instance terminals or pins. If more than one member is being used, they must be attached to the same net.

Supporting Applications

Electrically Aware Design (EAD) Flow

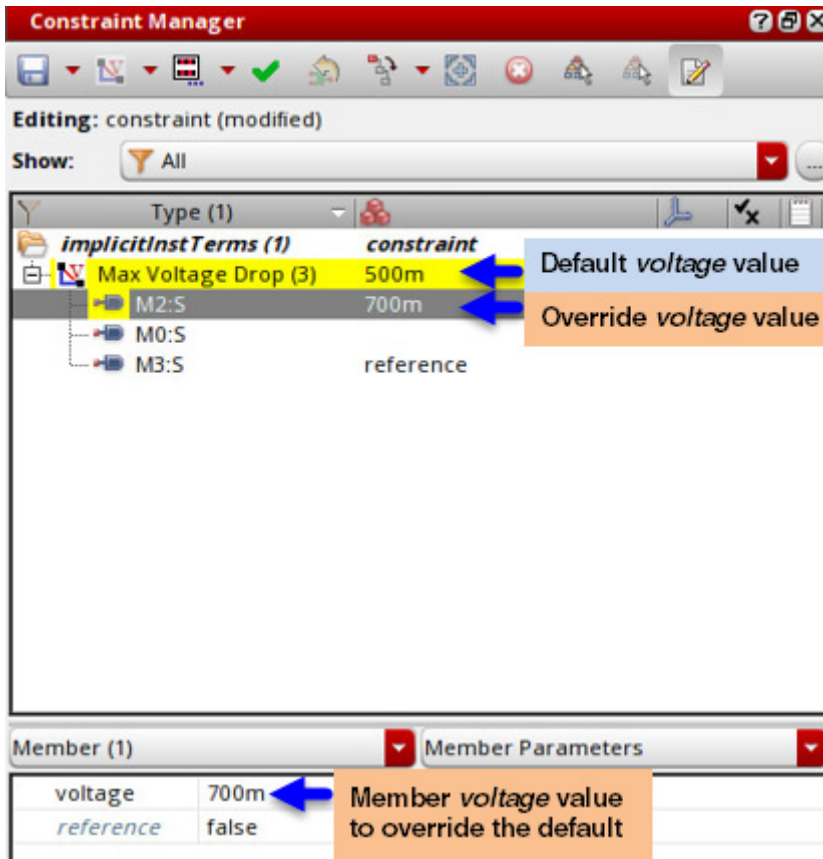
Overriding Voltage Drop of a Constraint Member

The *voltage* parameter on the constraint is the default voltage drop for all member instance terminals/pins with respect to the reference instance terminal/pin.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

The value set for the constraint *voltage* parameter can be overridden on individual instance terminal/pin members by setting the member *voltage* parameter, as shown below.



When you set the member *voltage* parameter to zero, the *voltage* value specified for the constraint is used. In addition, the member reference instance terminal must have its voltage set to zero.

Modgen

Definition The Modgen constraint defines a two-dimensional arrangement of instances and internal routing for a set of devices.

Note: When you select any dummies from the list displayed on expanding a Modgen constraint in the Constraint Manager assistant, the selected dummies get cross-highlighted in the Modgen canvas editor. Similarly, this useful feature of cross-probing dummies, highlights any selections made on the Modgen canvas editor back in the Constraint Manager assistant.

Parameters In addition to the parameters *Name*, *Enabled*, and *Notes*, you can set the following parameters:

Note: All member parameters are only valid for instance and LCV members unless noted otherwise.

■ Nb of Rows

Type: integer

Default: 0

Valid Values: ≥ 0

On Constraint: Y

On Member: N

Display: Y

User Units: N

■ Nb of Cols

Type: integer

Default: 0

Valid Values: ≥ 0

On Constraint: Y

On Member: N

Display: Y

User Units: N

■ router

Type: enum

Default: none

Valid Values: none, pinToTrunk

On Constraint: Y

On Member: N

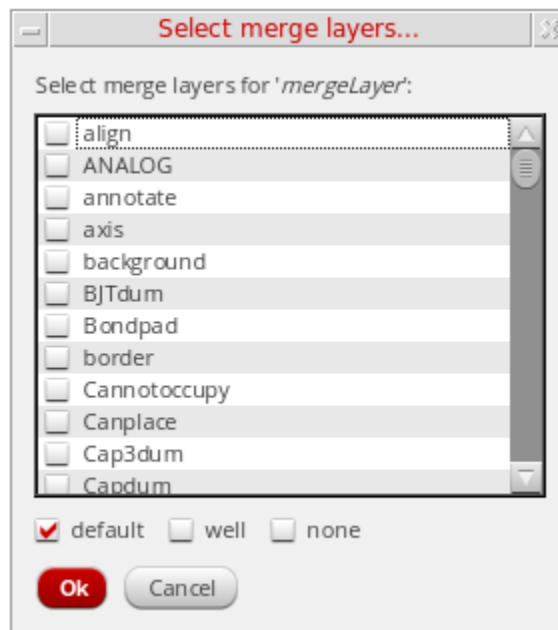
Display: Y

User Units: N

■ Merge Layers

Specify the Merge Layer, for example *default*, *none*, *well*, *Row*, or *Group*.

In the constraint parameters pane, if you click the value corresponding to Merge Layers, the *Select merge layers...* dialog box is displayed.



This dialog box provides the same functionality as explained in the [Specifying Merge Layers for Generated Modgen](#) section.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Note: In addition to the parameters described above, you can view the *Pattern* parameter. By default, it is hidden for a Modgen constraint. To make it visible again, you need to set the attribute `visibleInEditor` to `true` in the `config.xml` file as shown below.

```
<ConstraintType>
  <Name>modgen</Name>
  <GUIName menu="Placement">Modgen</GUIName>
  ....
  <Param>
    <Name>pattern</Name>
    <GUIName summary="ValOnly"
visibleInEditor="true">Pattern</GUIName>
  </Param>
  ....
</ConstraintType>
```

If the *Pattern* parameter is made visible, it has the following associated attributes:

Type: string

Default: Null (specified using "")

Valid Values: (AA BB ...)

On Constraint: Y

On Member: N

Display: Y

User Units: N

Note: Remembers last pattern entered.

Members

One or more instances and nets (for dummies and body contacts), and/or LCVs (for dummies). MFactor means the same schematic instance can be a member more than one time.

Instances may not appear in more than one Modgen constraint.

Note: A Modgen constraint can also be a *constraint container*. That is, it can contain its own members and also be used as a member itself for other constraints.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Supporting Applications

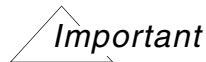
Modgen Editor

Note: The figGroup created by modgen is preserved in Constraint Aware Editing, Analog Auto Placer, Digital Placer, and the Floorplanner.

Net Class

Definition

The Net Class constraint is used to create groups of nets that have a specific set of constraints (typically process rule overrides) on or between its members, or with respect to other nets in the design. A net class constraint can contain nets.



Creation of nested Net Class constraints is not allowed from IC6.1.7 ISR5 and ICADV12.2 ISR5 onwards. If you have any existing nested Net Class constraints in your design and you try to update them, a warning is displayed in the CIW. To be able to update such constraints, you need to convert one or all nested Net Class constraints in a cache to corresponding Net Class Hier Group constraints by using the ciConvertNestedNetClassToNetClassHierGroup SKILL function.

The Process Rule Editor can be used to apply process constraints on:

- all members of a net class (default)
- between net members of the net class (within Group)
- between all net members of the net class and other nets in a design (group to outside)
- between members of two or more other member net classes (group to group)

Note: In the last case listed above, the net class can contain other net classes as its members.

This group of nets is not useful for any supporting applications until process rules have been added to the group. Collecting the nets in a net class does not have any special meaning for the router. When process rules are applied to the net class, the router respects them (as detailed here).

The `pinProximity` parameter is used to place all pins, on the listed nets, close to each other.

Note: Process rules can be added to the net class to define more parameters.

Parameters

In addition to the parameters *Name*, *Enabled*, and *Notes*, you can set the following parameters:

■ **pinProximity**

Type: bool

Default: false

Valid Values: t

On Constraint: N

On Member: Y

Display: N

User Units: Y

■ **Default Group (Constraint Group)**

Contains the following parameters: *Top Layer*, *Bottom Layer*, *Purposes*, *Vias Allowed*, *Widths Minimum*, *Cuts Minimum*, *Spacing*.

The above parameters are used to reflect the contents of a Net Class's default constraint group. Constraints may be placed here to locally override foundry rules obtained through standard lookup. Adding a constraint parameter to a Net Class's default constraint group is equivalent to adding the same constraint in each net member's default constraint group.

If there are similar constraints in the net's default constraint group, and the Net Class default constraint group, then the net's default constraint group parameter settings will have precedence. A typical use here would be to route a set of nets with a specific wider width or a net with a specific number of via cuts, or to modify the valid layers for all nets in a netclass.

Note: The *Default Group* parameters will only be available when the members of a Net Class constraint are nets (that is, not a nested net class).

In addition to the parameters listed above, *Default Group* has the *Wire Types* parameter that can be used to define a set of wire types for each layer. For more information, see [Assigning a Wire Type to a Net](#).

For more information on the Default Group, see [The Process Rule Editor](#).

For more information on customizing constraint groups, see [Customizing a Constraint Group](#).

■ Within Group (Constraint Group)

The *Within Group* parameters help to define the process rules for the net members in a Net Class constraint. These parameters include *Spacing* and *Tolerance*.

In addition to these two parameters, you can define the *Same Mask* parameter to ensure all shapes on a given layer for a net are on the same mask. This parameter allows you to specify the color of the nets in the net class. This can also have a hard or a soft attribute.

Note: The *Same Mask* parameter is equivalent to using the `msSameMask` constraint. If you reset the value of this constraint, the color of the nets should be reset accordingly.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

The *Same Mask* drop-down list has two options:

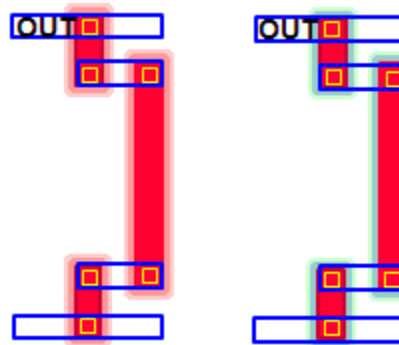
- ❑ `true`: This option ensures that the same mask is used for all nets.
- ❑ `false`: This option allows you to use the different mask.

Constraint (1) ▼ All Parameters ▼	
Name	Constr_0
Owner	neoexamples1....
Enabled	true
Status	not checked
Notes	
<i>pinProximity</i>	false
⊕ Default Group	Empty
⊖ Within Group	Empty
⊕ Spacing	(various)
<i>Tolerance</i>	20
<i>Same Mask</i>	false
⊕ Group to Outside Group	Empty
hierarchicalScope	local

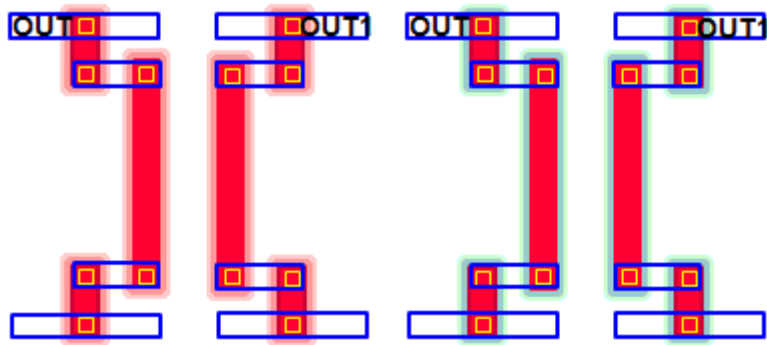
Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

The layout below shows the implementation of a Net Class constraint on a single net when *Same Mask* is set to `true`.



The layout below shows the implementation of a Net Class constraint on multiple nets when *Same Mask* is set to `true`.



The *Within Group* parameters are available only when the members of a net class are nets (that is, not a nested net class).

For more information on the *Within Group* parameters, see [The Process Rule Editor](#).

■ Group to Outside (Constraint Group)

The *Group to Outside* parameter includes specification of only the *Spacing* parameters, which define the spacing between the net class members and everything else in the design.

These parameters can be used to enter the process rules between the nets in a net class constraint and those nets that are not members of that constraint.

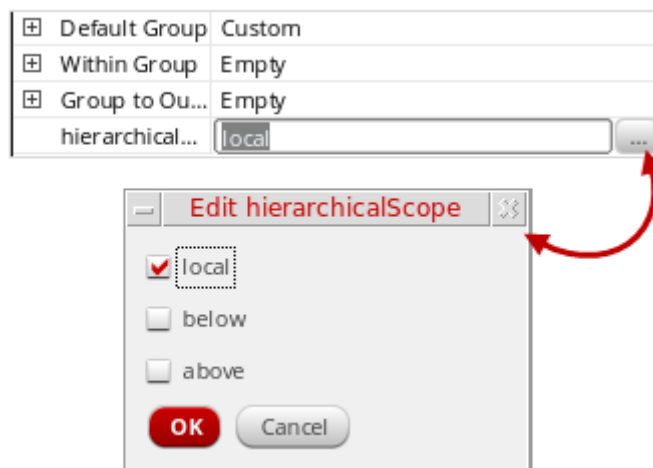
It is available only when the members of a net class are nets (that is, not a nested net class).

For more information on Group to Outside, see [The Process Rule Editor](#).

See also the figure at the end of this description.

■ hierarchicalScope

Specifies the current hierarchical scope of the constraint, that is, whether *local*, *below*, or *above* the current cell level. To edit this parameter, click the displayed value and then click the button displayed on the right. This displays the Edit hierarchicalScope form with the *local*, *below*, and *above* check boxes available for selection.



Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Members

- A Net Class constraint that contains nets.

The Default, Within Group and Group to Outside parameters will be available in this case.

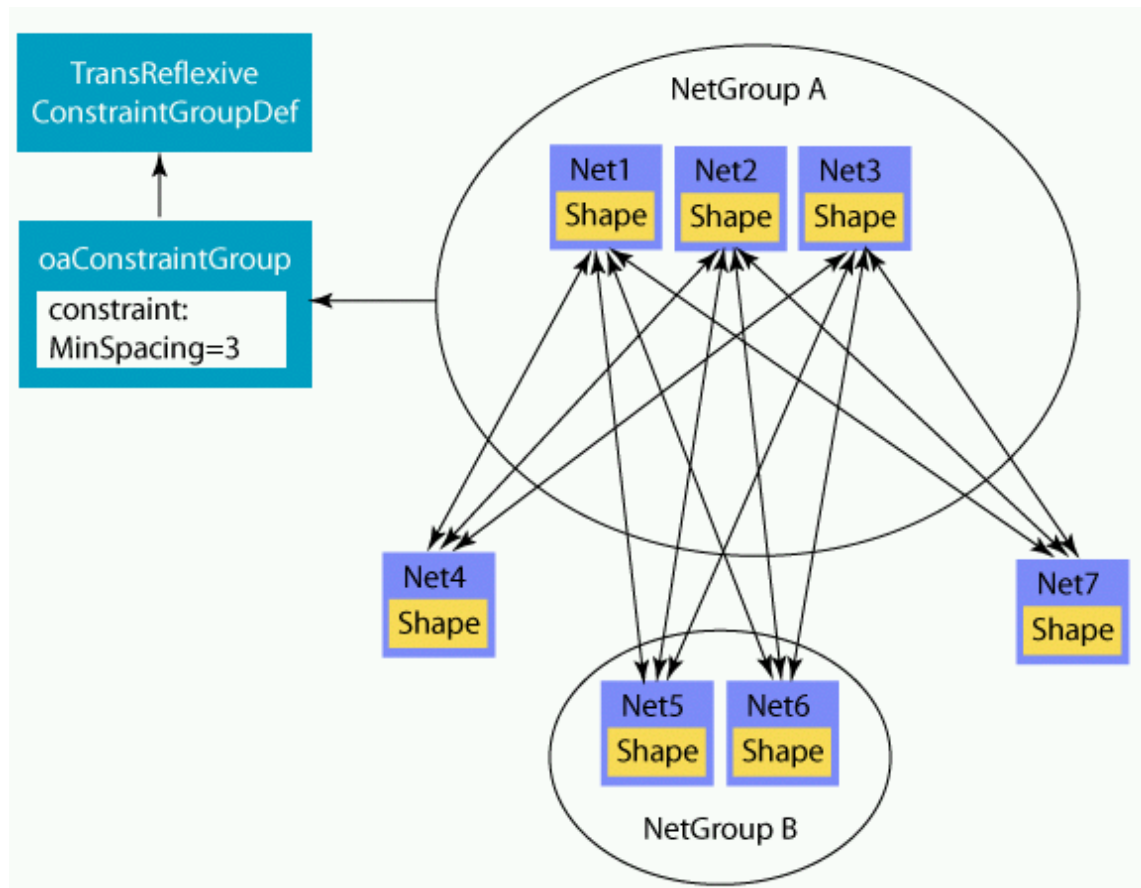
- A Net Class constraint that contains other Net Class constraints, that is a nested net class.

The only available parameter here will be Group to Group. The Group to Group parameter details process rules that are enforced between any net in a sub-group and all members in all other sub-groups.

Supporting Applications

- Virtuoso Layout Suite XL (Constraint Aware Editing)
- Virtuoso Layout Suite XL (Create Wire Editing)
- Virtuoso Space-based Router

The following figure shows constraints applied between nets in a net class and all other nets:



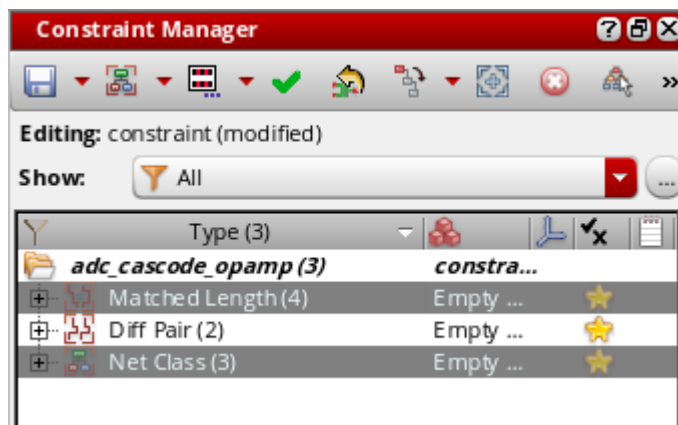
Net Class Group

Definition

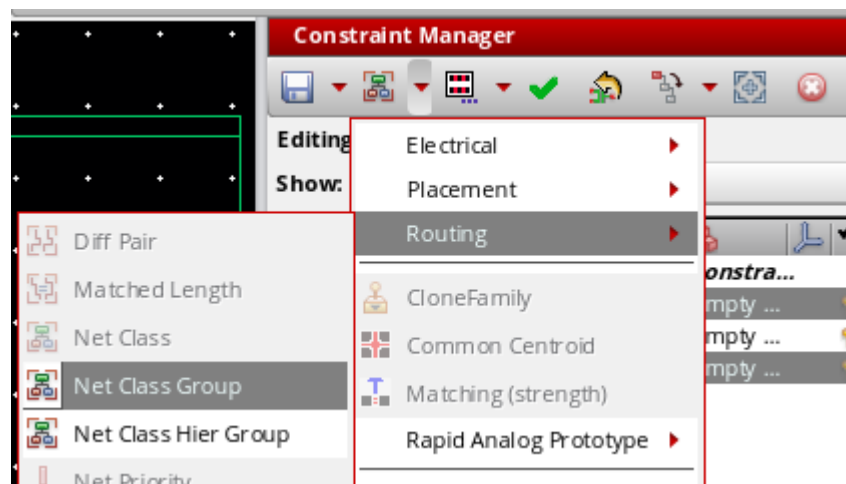
The Net Class Group constraint is used to create a group of one or more members belonging to the following routing-related constraint types: Diff Pair, Bus, Matched Length, Net Class, Strand, and Symmetry.

To create a Net Class Group constraint,

1. Select the required constraint types in the *Constraint Browser*, as shown below.



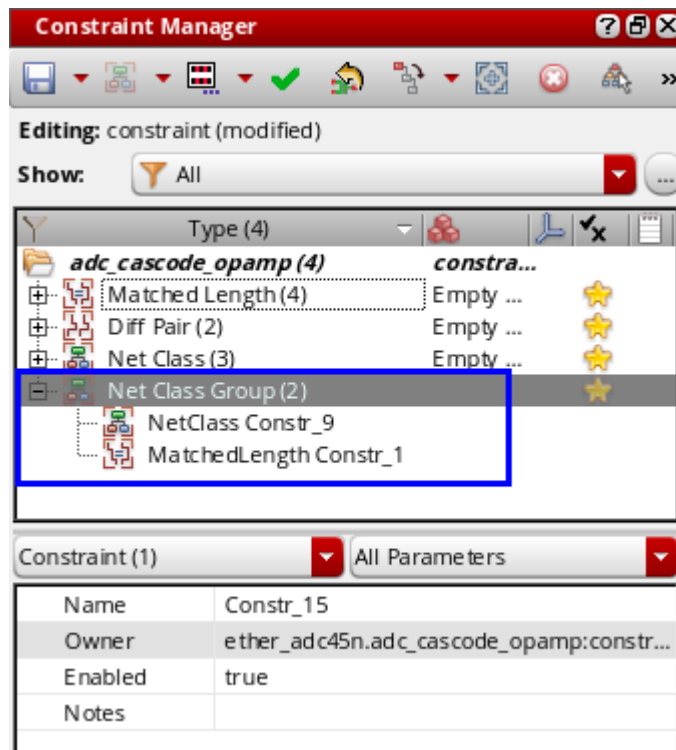
2. Click the *Constraint Creation* menu and place the pointer on *Routing* to expand and view the related constraint types.



Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

3. Select *Net Class Group*. You can now see the newly-created constraint listed in the *Constraint Browser*.



Important

See also [Important Points to Note During Net Class Group Constraint Creation](#).



Video

For a video demonstration, see [Creating Net Class Hier Group Constraints](#) on Cadence Online Support.

Parameters

You can set the following parameters: *Name*, *Enabled*, and *Notes*.

Members

One or more constraints of the following types: Diff Pair, Bus, Matched Length, Net Class, Strand, and Symmetry (containing only nets as members).

Virtuoso Unified Custom Constraints User Guide

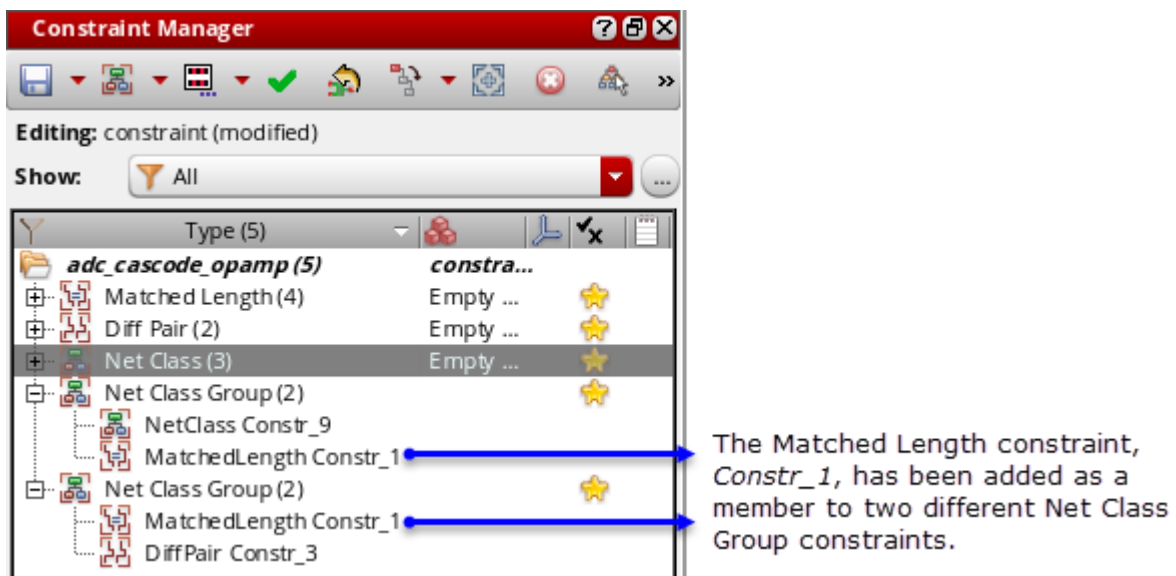
Default Constraint Types

Supporting Applications

- Virtuoso Layout Suite XL (Constraint Aware Editing)
- Virtuoso Layout Suite XL (Create Wire Editing)
- Virtuoso Space-based Router

Important Points to Note During Net Class Group Constraint Creation

- A member of one Net Class Group constraint can also be a member of another Net Class Group constraint, as shown below.



- You can also add new members to an existing Net Class Group constraint by drag and drop.
- The same member can be added to multiple Net Class Group constraint, as shown below. As a result, any combination of members can be selected to create a Net Class Group constraint.



Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

- For creation of a Net Class Group constraint, no checks are performed for legal or illegal members. However, when you create a Net Class Hier Group constraint, such checks are performed to ensure that there are no spacing-related conflicts. For detailed information, refer to Important Points to Note During Net Class Hier Group Constraint Creation section.
- A Net Class Group constraint cannot contain nested Net Class constraints.
- Symmetry constraints containing instances and modgens cannot be added as members to a Net Class Group constraint. Symmetry constraints with only nets are allowed.

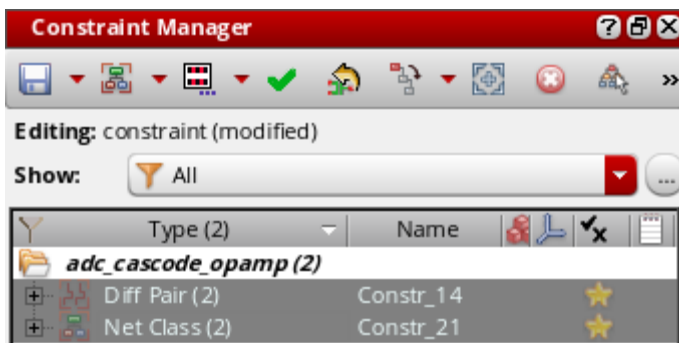
Net Class Hier Group

Definition

The Net Class Hier Group constraint is used to create a group of one or more members belonging to the following routing-related constraint types: Diff Pair, Bus, Matched Length, Net Class, Net Class Group, Strand, and Symmetry. A Net Class Hier Group constraint has no limitation to the number of members that can be added to it.

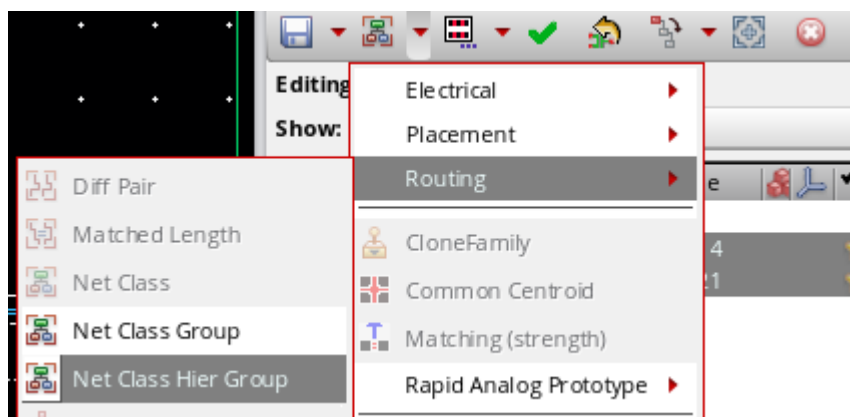
To create a Net Class Hier Group constraint,

1. Select the required constraint types in the *Constraint Browser*, as shown below.



You can create a Net Class Hier Group without creating a Net Class Group constraint. For example, in the figure above, two constraints – Diff Pair and Net Class – have been selected to group and space out by creating a Net Class Hier Group constraint.

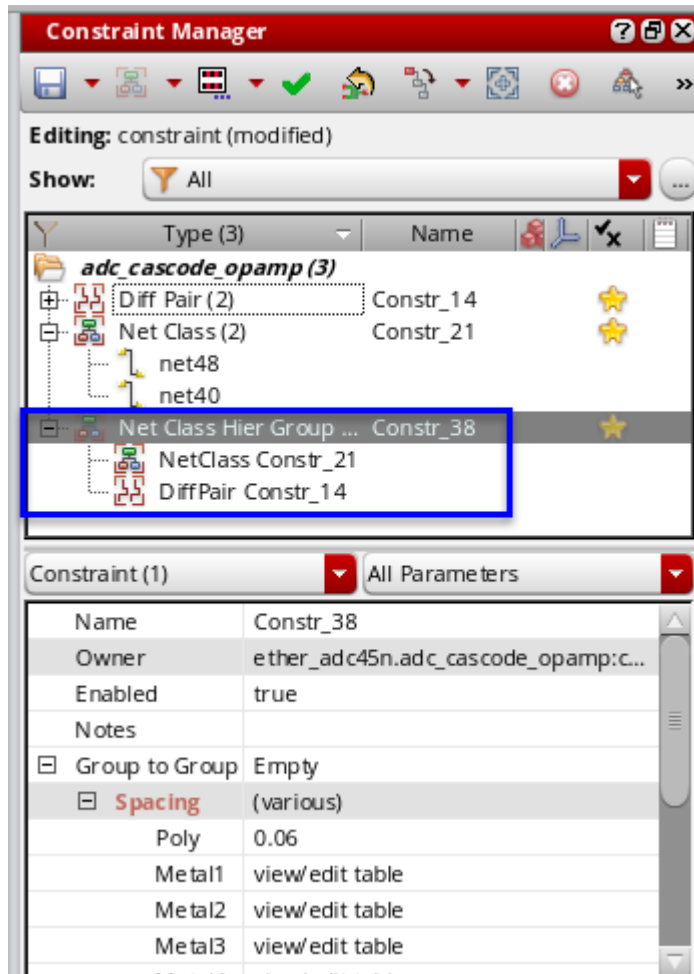
2. Click the *Constraint Creation* menu and place the pointer on *Routing* to expand and view the related constraint types.



Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

3. Select *Net Class Hier Group*. You can now see the newly-created constraint listed in the *Constraint Browser*.



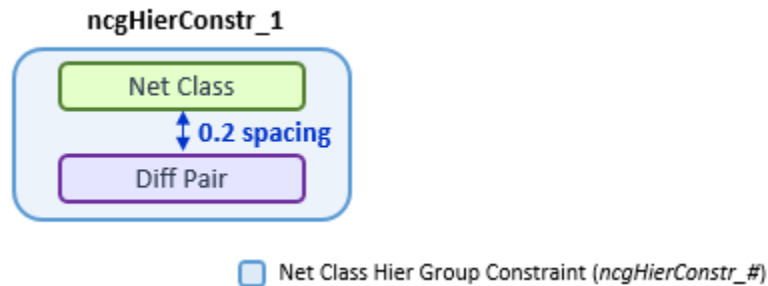
Tip

See also [Important Points to Note During Net Class Hier Group Constraint Creation.](#)

Creation of a Net Class Hier Group constraint is useful when there is a need to define spacing between a set of different constraint types. The defined spacing applies to each member of a Net Class Hier Group constraint if it does not contains a Net Class Group constraint, as shown below.

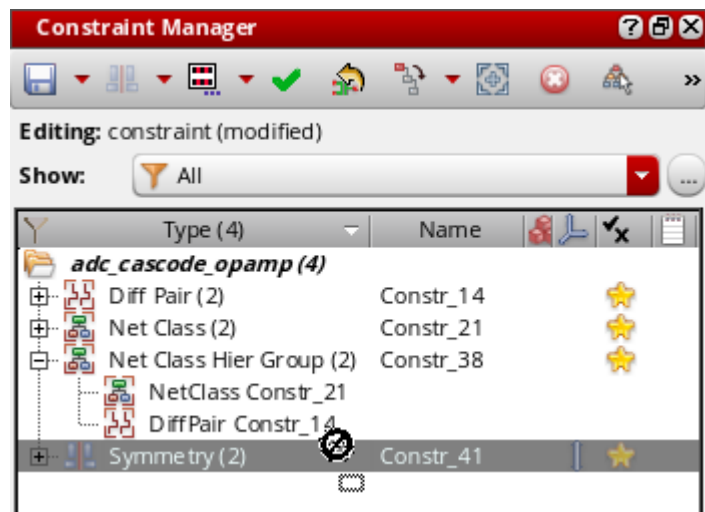
Virtuoso Unified Custom Constraints User Guide

Default Constraint Types



If the Net Class Hier Group constraint contains a Net Class Group constraint, the members of the Net Class Group constraint are treated as a single entity. Therefore, the spacing applies to the complete Net Class Group and not its individual members.

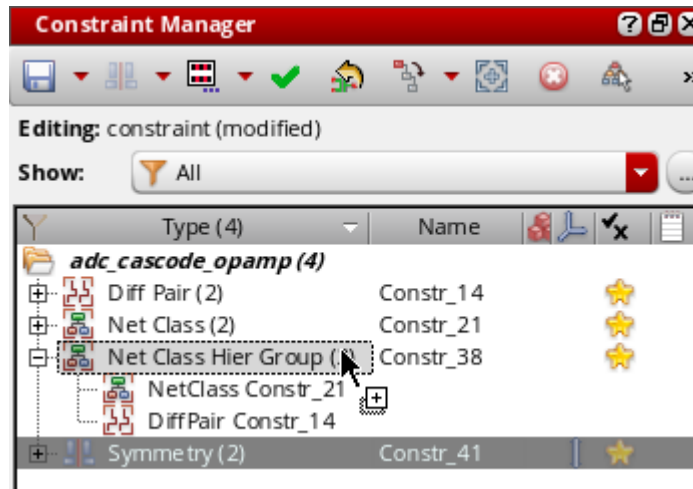
You can also define spacing between the existing member constraints of a Net Class Hier Group and another constraint that is not a member already. For example, in the illustration above, you added Diff Pair (Constr_14) and Net Class (Constr_21) to a Net Class Hier Group (Constr_38). Now, you want to define the spacing between this Net Class Hier Group and a Symmetry constraint, Constr_41. To do this, select the Symmetry constraint and drag it towards the Net Class Hier Group constraint, as shown below.



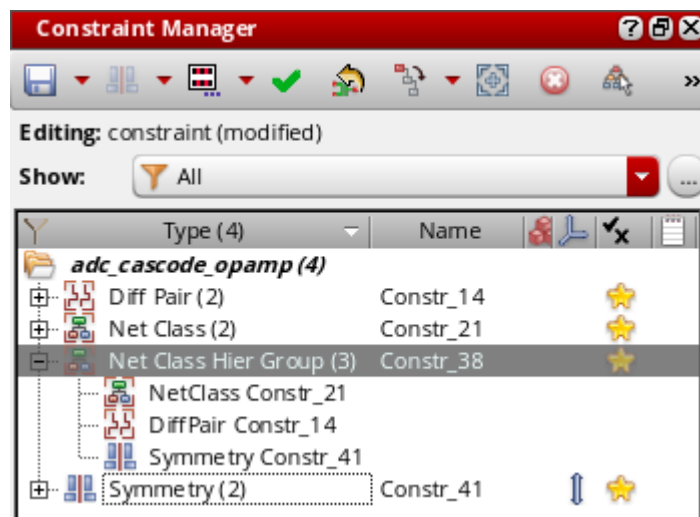
Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

When the pointer starts to show a plus (+) sign on the Net Class Hier Group constraint, release the mouse button to drop and add to it the Symmetry constraint.



The Net Class Hier Group constraint starts to reflect the new addition, as shown below.



Virtuoso Unified Custom Constraints User Guide

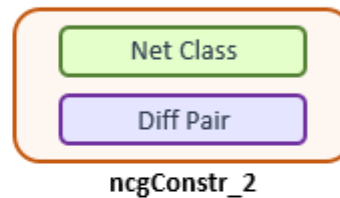
Default Constraint Types

If you want to space out a constraint from a set of constraints, first create a Net Class Group constraint on the combined set of constraints and then create a Net Class Hier Group constraint with that Net Class Group constraint and the other constraint as members, as shown below.

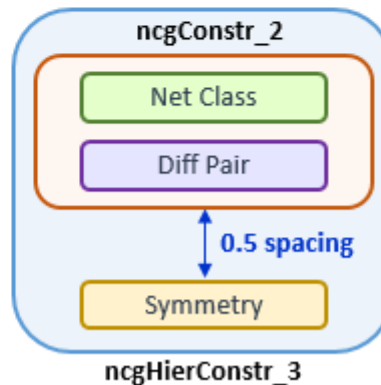
- 1 Suppose you need to space out the Symmetry constraint from two other constraints – Net Class and Diff Pair



- 2 Then, create a Net Class Group constraint, *ncgConstr_2*, on Net Class and Diff Pair.



- 3 Now, create a Net Class Hier Group constraint, *ncgHierConstr_3*, on *ncgConstr_2* and Symmetry with a spacing of 0.5.



- Net Class Group Constraint (*ncgConstr_#*)
- Net Class Hier Group Constraint (*ncgHierConstr_#*)

Video

For a video demonstration of the process discussed above, see [Creating Net Class Hier Group Constraints](#) on Cadence Online Support.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Parameters

In addition to the parameters *Name*, *Enabled*, and *Notes*, you can set the following parameter:

■ Group to Group (Constraint Group)

It contains the *Spacing* parameter that defines the minimum spacing between the members of the Net Class Hier Group constraint. Spacing is defined per layer in user units.

Members

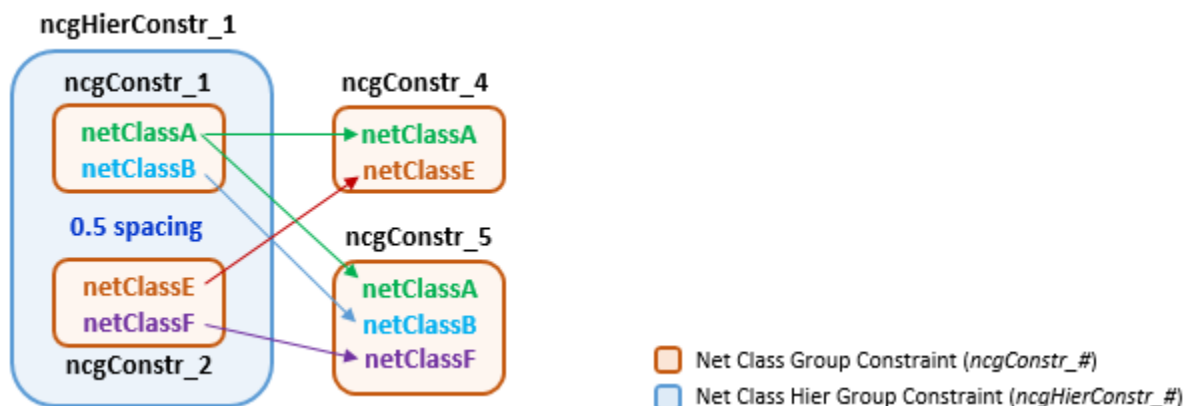
One or more constraints of the following types: Diff Pair, Bus, Matched Length, Net Class, Net Class Group, Strand, and Symmetry (containing only nets as members).

Supporting Applications

- Virtuoso Layout Suite XL (Constraint Aware Editing)
- Virtuoso Layout Suite XL (Create Wire Editing)
- Virtuoso Space-based Router

Important Points to Note During Net Class Hier Group Constraint Creation

- A Net Class Hier Group constraint can contain any number of members; there is no restriction. However, consistency checks are performed to avoid spacing-related conflicts between the members of the Net Class Hier Group constraint especially when it contains a Net Class Group constraint.
- A member of the supported constraint types can exist in more than one Net Class Group and Net Class Hier Group constraint, as shown below.



- A Net Class or Net Class Group constraint cannot be a member of multiple Net Class Hier Group constraints because difference in spacing might create ambiguity. For example, there are four net classes – A, B, C, and D. There can be two Net Class Hier

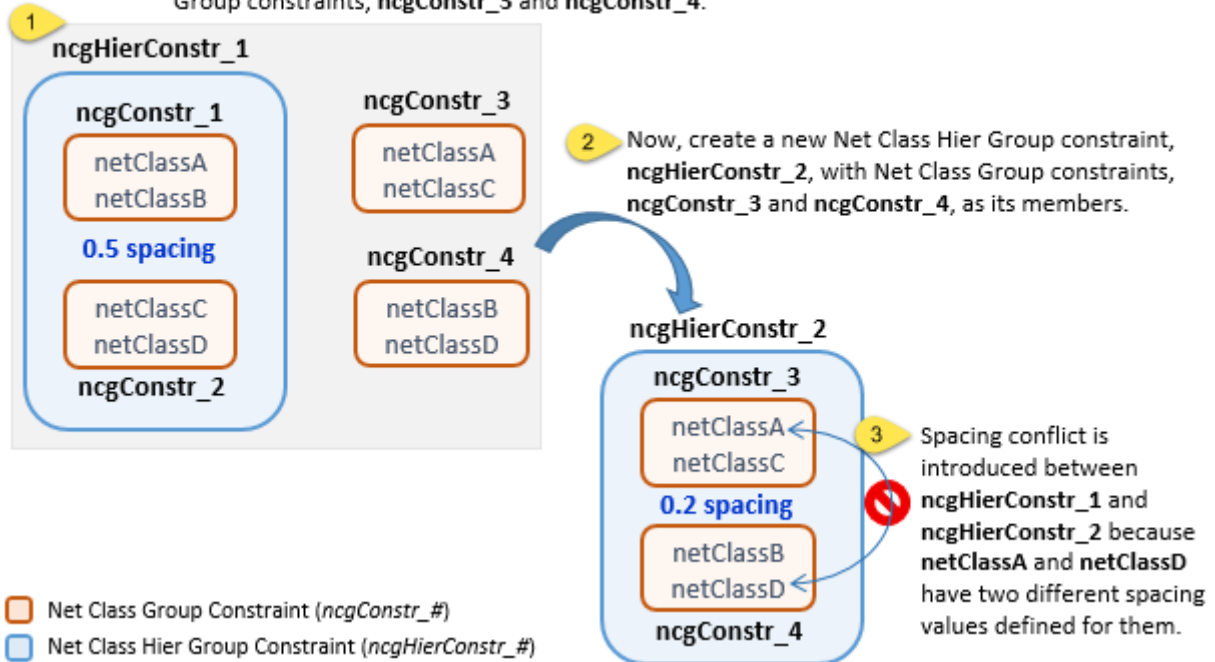
Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Group constraints on (A,B) and (C,D) with certain spacing values. Now, if a new Net Class Hier Group is created on (A,B,C,D) with a different spacing value, it might cause a conflict with the originally defined spacing between (A,B) and (C,D).

The figures below illustrate some spacing-related conflicts:

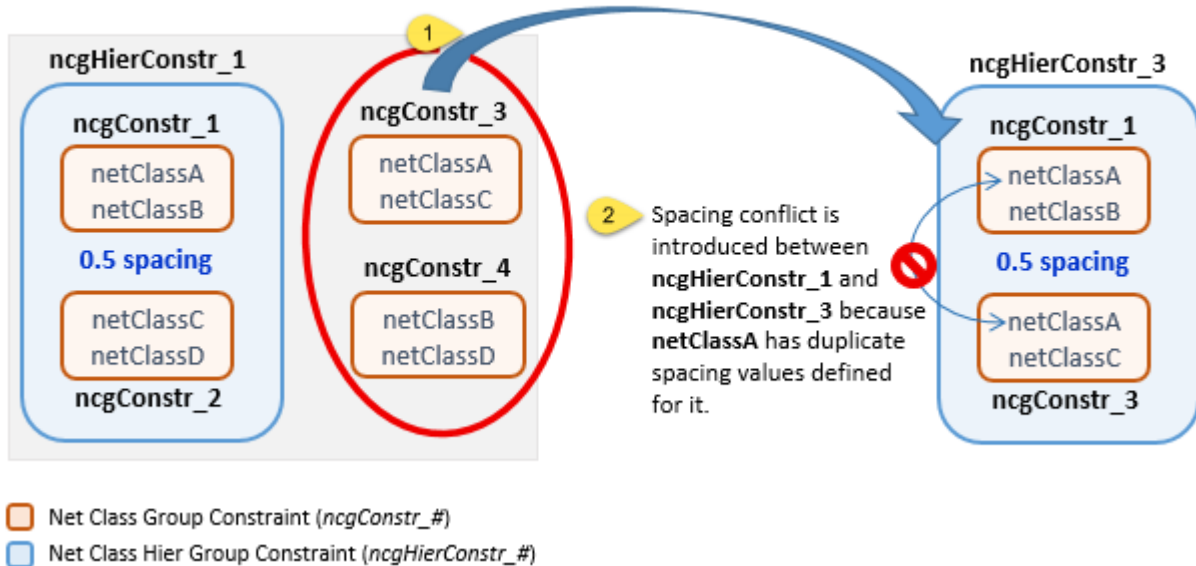
Scenario 1: You created a Net Class Hier Group constraint, **ncgHierConstr_1**, with two member Net Class Group constraints, **ncgConstr_1** and **ncgConstr_2**. In addition, there existed two other Net Class Group constraints, **ncgConstr_3** and **ncgConstr_4**.



Virtuoso Unified Custom Constraints User Guide

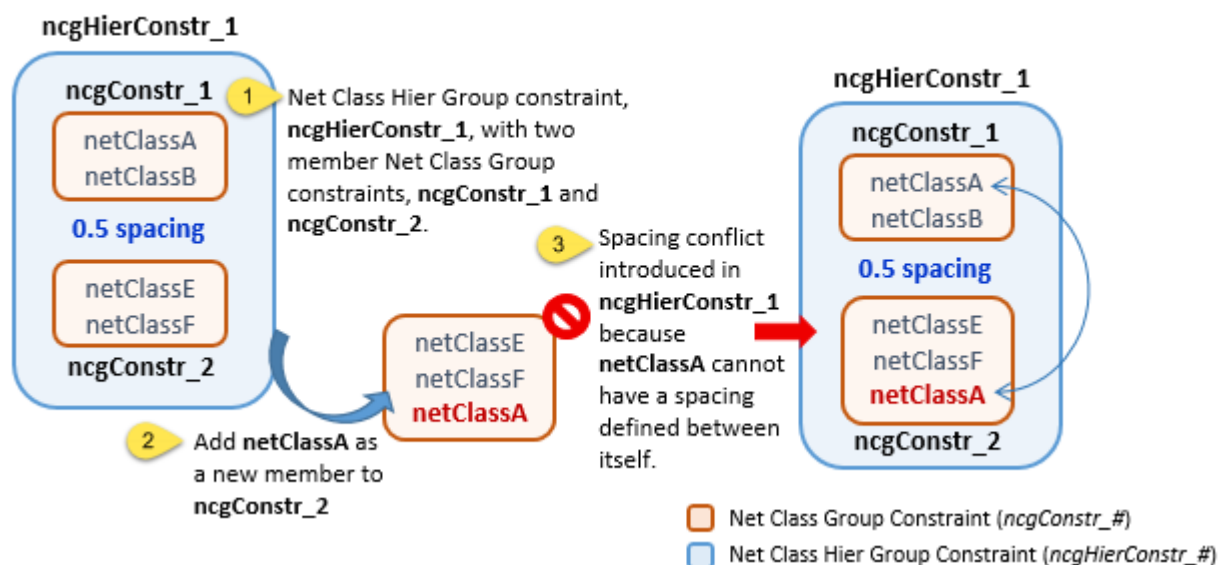
Default Constraint Types

Scenario 2: Continuing with the example above, say you created a new Net Class Hier Group constraint, **ncgHierConstr_3**, with Net Class Group constraints, **ncgConstr_1** and **ncgConstr_3**, as its members.



- When you drag and drop a new member to add to a Net Class Group constraint that is a member of a Net Class Hier Group constraint, checks are performed for spacing. If a spacing-related conflict is found between the Net Class Group's new member and the Net Class Hier Group's existing members, a warning is displayed in the CIW and the new member is not added to the selected Net Class Group Group constraint. The figures below illustrate scenarios of spacing-related conflicts in such condition:

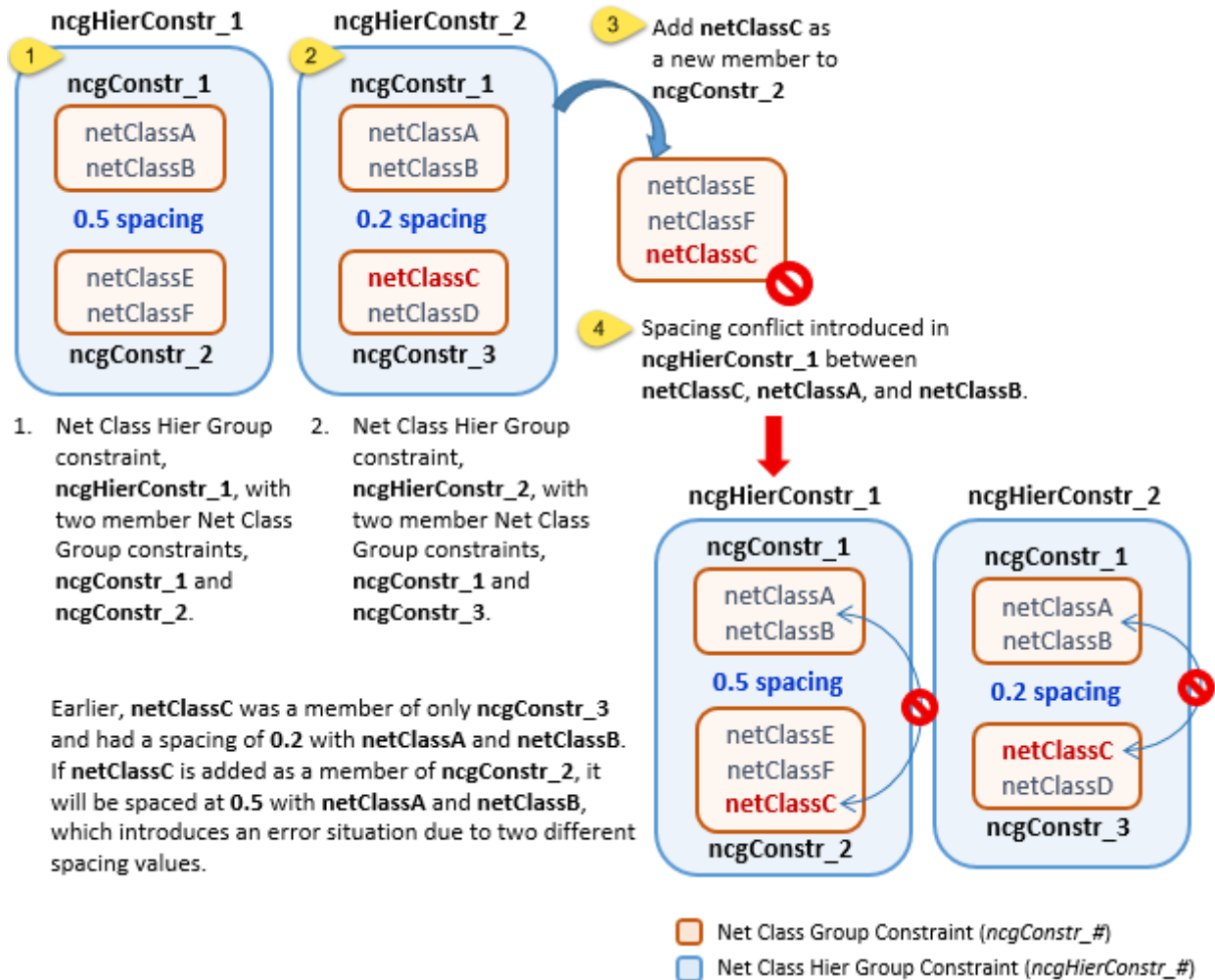
Scenario 1:



Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Scenario 2:

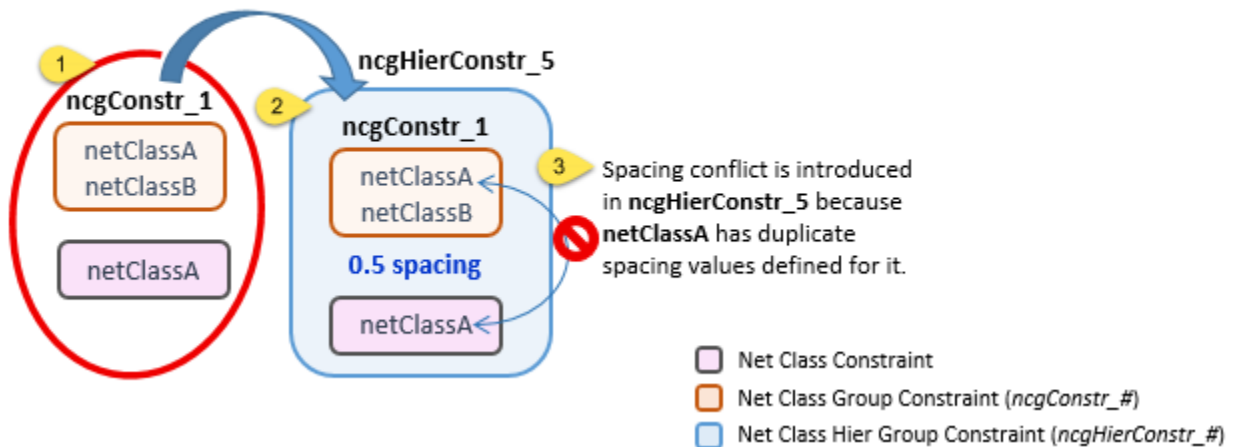


Virtuoso Unified Custom Constraints User Guide

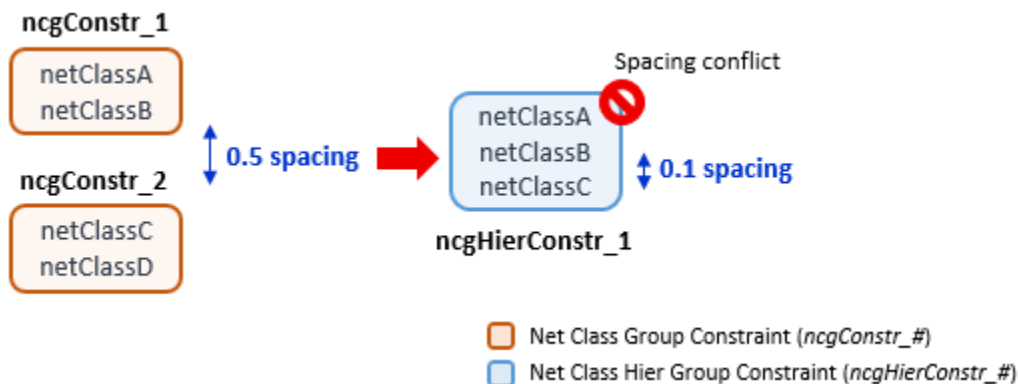
Default Constraint Types

- If a Net Class Group constraint exists with a specific set of members, you cannot create a Net Class Hier Group constraint using this Net Class Group constraint and its members individually, as shown below:

Create a Net Class Hier Group constraint, **ncgHierConstr_5**, with Net Class Group constraint, **ncgConstr_1**, and **netClassA** as its members.



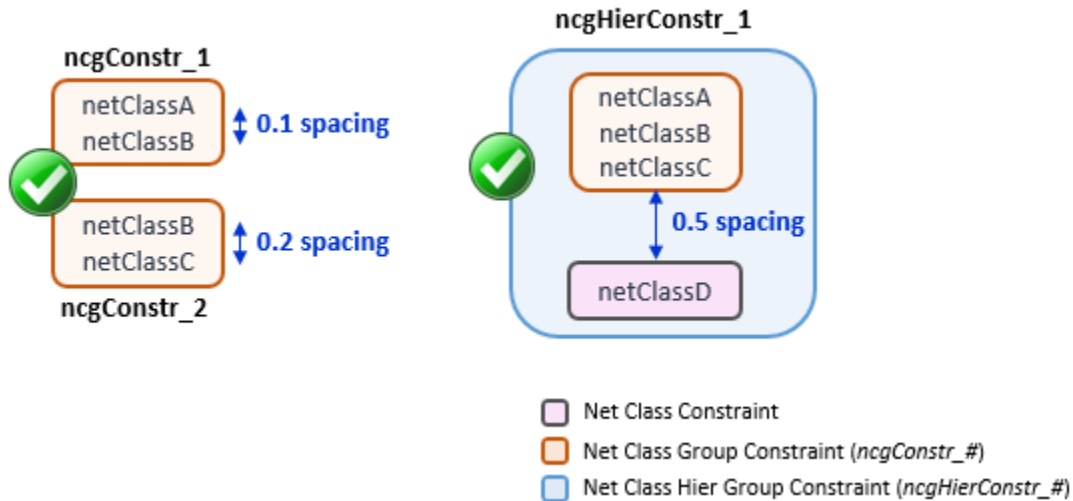
- Definition of spacing at the level of hierarchically-nested Net Class Group constraints is not allowed. As a result, a scenario as shown in the figure below generates an error:



Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

However, a scenario like the one depicted in the figure below is acceptable:



- A Net Class Hier Group constraint cannot contain nested Net Class constraints.
- Symmetry constraints containing instances and modgens cannot be added as members to a Net Class Group constraint. Symmetry constraints with only nets are allowed.

See also the [Important Points to Note During Net Class Group Constraint Creation](#) section.

Net Priority

Definition

The Net Priority constraint defines the order of priority when routing a net. The higher the value, the higher the priority.

A higher priority implies that the net is more critical and therefore needs to be routed more optimally (in terms of wire length as well as any user defined constraint on it), even if that makes routing of the lower priority nets less optimal.

When a placement tool supports the net priority constraint, it will place together the devices that are inter-connected with high priority nets to minimize the length of these nets.

When you attempt to create a Net Priority constraint in the *Constraint Manager*, the Create Priority window will be displayed prompting you to enter a *Priority* for the constraint being created. Do not enter a value of 0 here, as this will be rejected as an invalid value and the constraint will not be created. The valid value range is -128, 128.

Note: If you create a Net Priority constraint using the *Create Default Constraints* option in the *Circuit Prospector* assistant, a default priority value of 64 will be set for non-supply nets.

Important

The net priority constraint does not support parameter notes and history (timestamp) tracking (see [Constraint History](#)). Any Net Priority constraints that are out-of-context will be purged when the cache is closed.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Parameters

■ **priority**

Type: Int

Default: 64

Valid Values: (-128, 128) except 0

On Constraint: Y

On Member: N

Display: Y

User Units: N

Members

One or more nets.

Supporting Applications

- Virtuoso Space-based Router
- Virtuoso Analog Auto Placer
- Virtuoso Floorplanner
- Constraint Aware Editing
- Virtuoso Custom Digital Placer

Note: The net priority constraint cannot be stored as a template as it is an Open Access object attribute and does not have independent OA storage. For more information, see [Constraint Template Storage](#) in the [Virtuoso Unified Custom Constraints Configuration Guide](#).

Orientation

- Definition** The Orientation constraint defines the allowed orientations for the members of the constraint via its parameter Orientations. More than one orientation is allowed.
- Parameters** In addition to the parameters *Name*, *Enabled*, and *Notes*, you can set the following parameters:

■ Orientations

Type: enumSet

Valid Values: All, Vertical, Horizontal, Other

See also: [Pre-Set and Custom Orientations](#)

On Constraint: Y

On Member: N

Default: ALL (which means that all orientations are allowed)

Display: Y

User Units: N

Note: When a *Symmetry* constraint is set for two instances, the value of the *Orientations* parameter, for each instance, must include the legal pairs of values required by the *Symmetry* constraint.

For example, a *Mirror Symmetry* constraint with a **vertical** axis requires one of the following pairs of respective orientations:

- R0 and MY
- R90 and MXR90
- R180 and MX
- R270 and MYR90

For a *Mirror Symmetry* constraint with a **horizontal** axis, the legal pairs of respective orientations are:

- R0 and MX
- R90 and MYR90
- R180 and MY
- R270 and MXR90

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Members

One or more instances, modgen constraints or figGroups, or regular figGroups.

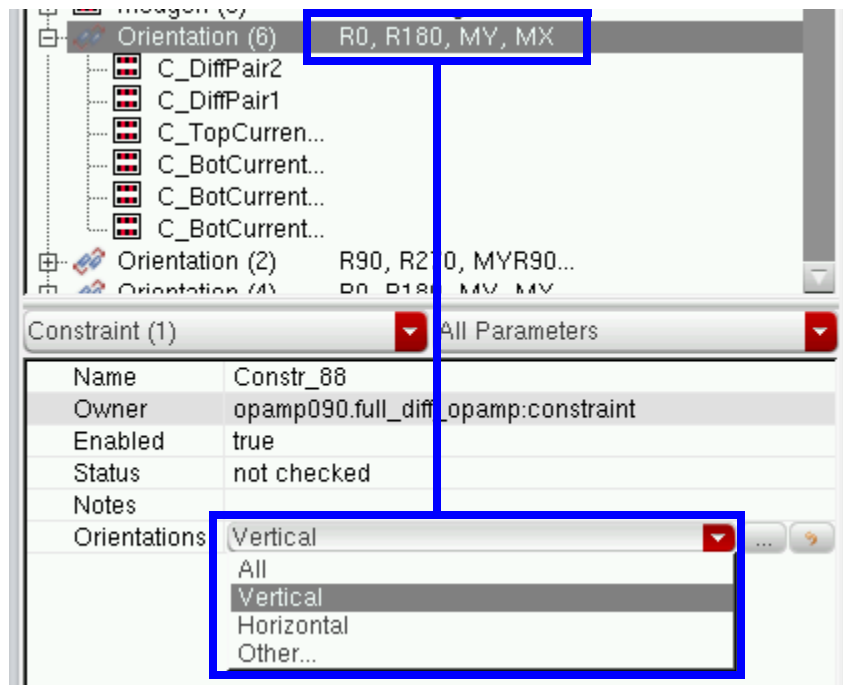
Note: In the case of an orientation and Matched Orientation constraint sharing two members, any implicit matched orientation specified in an Orientation constraint is required to be explicit in a Matched Orientation constraint.

Supporting Applications

- Virtuoso Custom Digital Placer
- Virtuoso Floorplanner
- Virtuoso Analog Auto Placer
- Virtuoso Layout Suite XL (Constraint Aware Editing)

Pre-Set and Custom Orientations

The *Orientations* parameter of the *Orientation* constraint provides four options to choose from: *All*, *Vertical*, *Horizontal*, and *Other*.

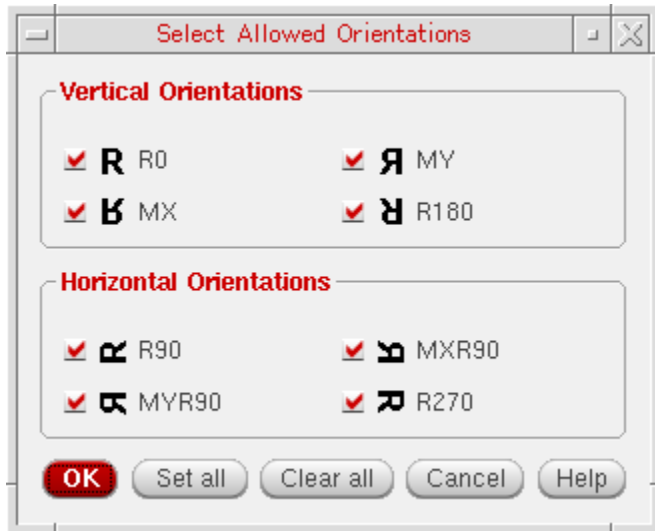


When you create the *Orientation* constraint, the *Orientations* parameter in the *Constraint Parameter Editor* pane shows *All* as selected by default.

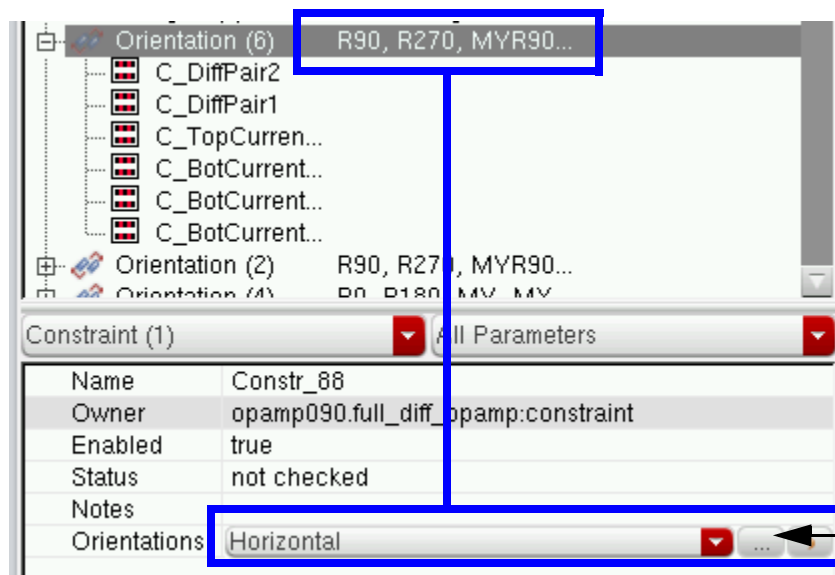
Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Clicking the  button adjacent to the *Orientations* parameter or selecting *Other...* from the drop-down list box opens the Select Allowed Orientations form. When *All* is selected and you click the button to open this form, all check boxes on the form appear selected, as shown in the figure below.



The *All* setting applies all possible orientations to the selected *Orientation* constraint, whereas selecting *Vertical* or *Horizontal* applies only the pre-set associated orientations to the constraint. For example, if you change the *Orientations* parameter selection in the *Constraint Parameter Editor* pane from *ALL* to *Horizontal*, the constraint orientation settings change automatically.

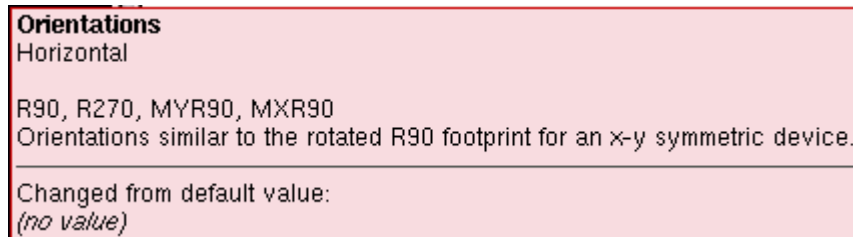


Click to select orientations

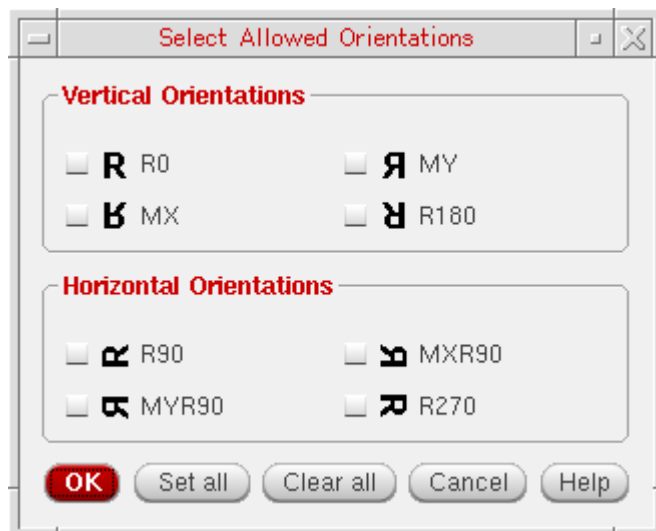
Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Note: When you move the cursor over the *Orientations* parameter value, a tooltip is displayed providing more information related to the current orientation setting.



In addition to being able to choose orientation settings of *All*, *Horizontal*, or *Vertical*, you can select *Other...* that displays the Select Allowed Orientations form with all check boxes deselected, as shown below.

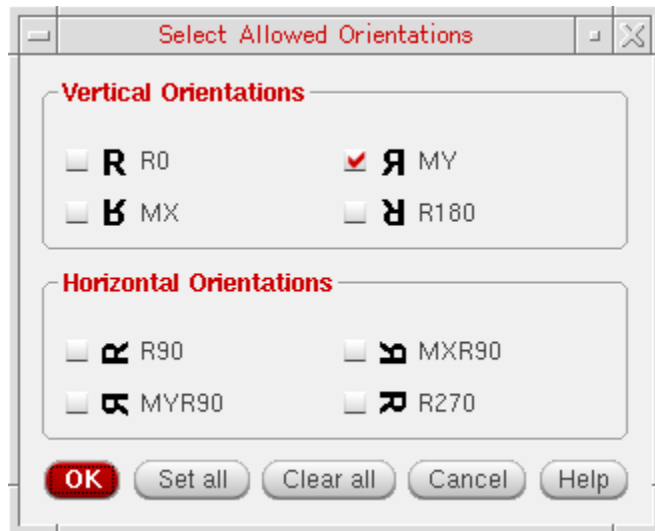


From here, you can choose your own, single or multiple, custom orientations.

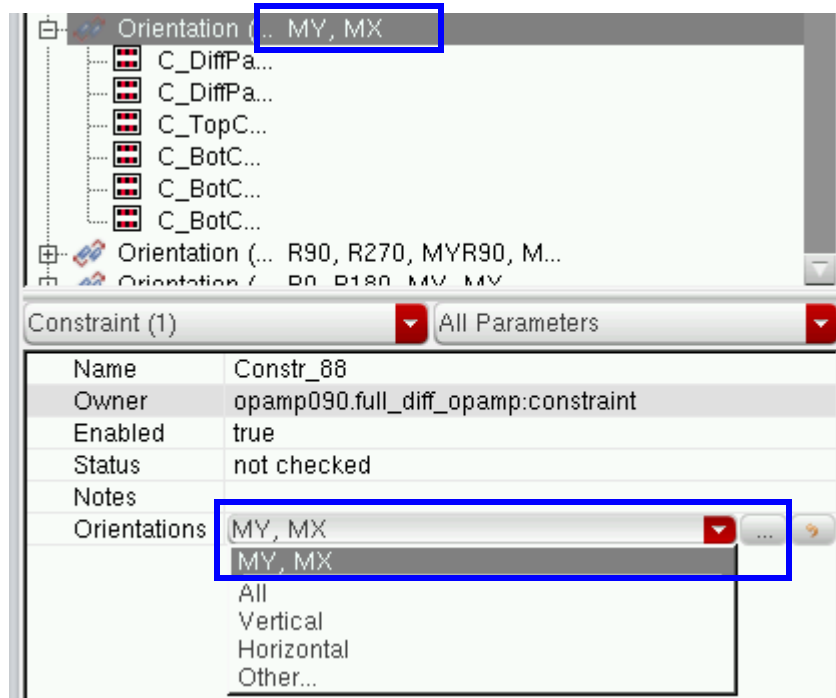
Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

When you open the Select Allowed Orientations form for an existing *Orientation* constraint, on initial display, the form appears pre-populate with the orientation settings that you saved for it last, as illustrated below.



Once you have selected the required orientation, the orientations are applied to the selected *Orientation* constraint and listed in the *Orientation*'s parameter drop-down list box for future use.



Different Orientation Options on the Select Allowed Orientations Form

By default, devices are considered as:

- ☐ Horizontal in the layout when their orientation matches one of the following: R0, R180, MY, or MX.
- ☐ Vertical in the layout when their orientation matches one of the following: R90, R270, MYR90, or MXR90.

When you open the Select Allowed Orientations form for a constraint that contains only 'default', the form has the orientation options shown in the following figure:

The screenshot shows a dialog box titled "Select Allowed Orientations". It is divided into two main sections. The first section, "Vertical Orientations", contains four checkboxes: ☐ R0, ☐ MY, ☐ MX, and ☐ R180. The second section, "Horizontal Orientations", contains four checkboxes: ☐ R90, ☐ MXR90, ☐ MYR90, and ☐ R270. At the bottom of the dialog, there are five buttons: "OK" (which is highlighted in red), "Set all", "Clear all", "Cancel", and "Help".

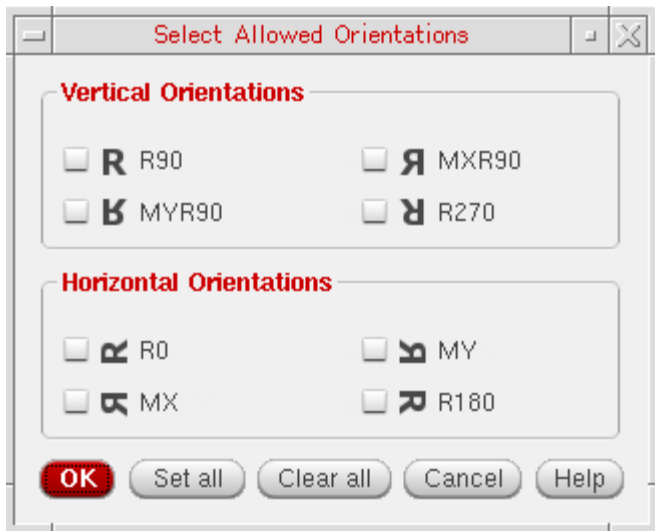
Devices registered as 'r0isHorizontal' are considered as:

- ☐ Horizontal in the layout when their orientation matches one of the following: R90, R270, MYR90, or MXR90.
- ☐ Vertical in the layout when their orientation matches one of the following: R0, R180, MY, or MX.

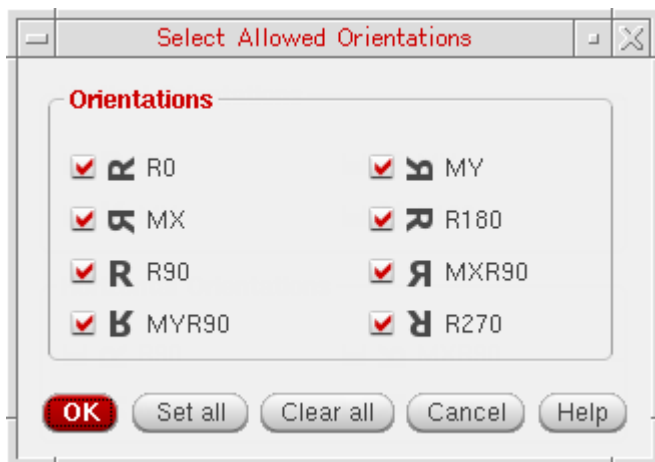
Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

When you open the Select Allowed Orientations form for a constraint that contains only device registered as 'r0isHorizontal', the form has the orientation options shown in the following figure:



When you open the Select Allowed Orientations form for a constraint that contains both types of devices (that is horizontal and vertical), the form has the orientation options shown in the following figure:



In this situation, the meaning of horizontal and vertical cannot be applied because the orientations are inverted for devices registered as r0isHorizontal compared to default device. The vertical and horizontal preset does not appear in the drop-down list box to edit the constraint value.

Placement Path

Definition The Placement Path constraint enables you to place instances of a given current path in a straight and ordered manner.

It identifies current paths in the active schematic cellview and creates a Placement Path constraint in the schematic Constraint Manager.

Parameters In addition to the parameters *Name*, *Enabled*, and *Notes*, you can set the following parameters:

■ **direction**

Type: enum

Default: auto

Valid Values: auto, horizontal, vertical

On Constraint: Y

On Member: N

Display: Y

User Units: N

■ **ordering**

Type: enum

Default: strict

Valid Values: strict, bulknet

On Constraint: Y

On Member: N

Display: Y

User Units: N

■ **exclusive**

Type: enum

Default: auto

Valid Values: true, false

On Constraint: Y

On Member: N

Display: Y

User Units: N

■ **Row**

Note: This parameter is enabled only when the Placement Path members are instances and the `direction = horizontal`, `ordering = strict`, and `exclusive = true`. It is not available for nested Placement Paths.

Type: float

Default: 1

Valid Values: any numerical value greater than 0

On Constraint: Y

On Member: N

Display: Y

User Units: N

Members

Placement Path constraints can be created on a single or multiple instances. The constraint member order determines the placement order for path members.

Note: A Placement Path constraint can contain a Placement Path as a member as long as that member contains only Placement Path objects.

The placer follows the *Direction* parameter to place the path members in the `horizontal` or `vertical` direction.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

- If set to `Auto`, the placer determines the direction automatically. Depending on the location of pins, one of the following placement direction is followed:
 - ❑ Top to Bottom (default)
 - ❑ Bottom to Top
 - ❑ Left to Right
 - ❑ Right to Left
- If no pin shape on the net is found, the direction is set to `vertical`.
- If `direction = horizontal`, `ordering = strict`, and `exclusive = true`, the Row parameter will also be enabled.

There are two implied priority levels of the Placement Path constraint and ordering is maintained in both cases:

- High: When Placement Path constraint has the following settings
 - ❑ `Exclusive = true` (Restrict non-path member instance insertion)
 - ❑ `Ordering = strict` (Restrict overlap in path order direction)
- Low: When Placement Path constraint has the following settings
 - ❑ `Exclusive = false` (Allows non-path member instance insertion)
 - ❑ `Ordering = bulknet` (Allows overlap in path order direction)
- Virtuoso Analog Auto Placer

Supporting Applications

Process Rule Overrides

Definition

Process Rule Overrides (PRO) can be used to override, on a given object, the generic foundry and design process rules with specific, more stringent, rules as defined in Constraint Groups (which are stored in designs or technology files and act as a collection of process constraints that are applicable to the objects in a design).

Note: A PRO constraint can be created for an entire design if nothing is currently selected (using the *Constraint Creation* pull-down in the *Constraint Manager* or the right-mouse button context menu). Here, the parameters will be the same as those available for PRO constraints created on nets.

The status (*passed* or *failed*) of Process Rule Override constraints will be updated when the *Verify – Design* command is run in VLS (the *Process Rules* tab is used to configure verification settings).

Note: When DRD (design rule driven) markers exist in a design, they should be read, processed, and then have their Constraint Status updated to ensure that their display setting is current for Process Rule Override constraints.

When a process rule violation exists, any process rule override constraint on a net, or on a net class associated with a marker created, will be marked as *failed*. All other process rule override constraints will be considered as *passed* if no violation is found on any shapes associated with their nets.

When PRO constraint is created or transferred, it immediately takes the name of its constraint group, which is generated as *CG_<number>*.

Note: The process rule override status is not persistent. It cannot be saved in the database.

See also:

- [Process Rules for Objects](#)
- [The Process Rule Editor](#)

Parameters

Note: See also [Process Rule Override Parameters](#) and the [Process Rule Editor \(PRE\)](#) for more information on the PRO parameters detailed below and also additional PRO parameters.

■ Default Group (Constraint Group)

It contains the following parameters: *Top Layer*, *Bottom Layer*, *Purposes*, *Vias Allowed*, *Widths Minimum*, *Cuts Minimum*, *Spacing*.

The *Default Group* parameters are used to reflect the contents of a net's default constraint group. Constraints can be placed here to locally override foundry rules obtained through standard lookup. Typical use here would be to route a net with a specific wider width or a net with a specific number of via cuts, or to modify the spacing value for a net or the valid layers.

The lookup order used to populate the *Default Group* parameters is default wire constraint group – design constraint group – foundry. You will not be able to edit values coming from the default wire constraint group. A tooltip for such values explicitly prompts you to edit them in PRE to override.

In addition to the parameters listed above, *Default Group* has the *Wire Types* parameter that can be used to define a set of wire types for each layer. For more information, see [Assigning a Wire Type to a Net](#).

See also [Custom Parameter Value \(for Process Rule Override Parameters\)](#).

For more information on customizing constraint groups, see [Customizing a Constraint Group](#).

■ Input Taper (Constraint Group)

■ Output Taper (Constraint Group)

Both these constraint group parameters contain the following parameters: *Halo*, *Top Layer*, *Bottom Layer*, *Purposes*, *Vias Allowed*, *Widths Minimum*, *Cuts Minimum*, *Spacing*.

The Input/Output Taper constraint group parameters are used to reflect the contents of the nets Input/Output taper default constraint group.

Constraints can be placed here to locally override rules obtained through standard lookup. Overridden rules in this location are only valid when set within a halo on a net near an Input/Output pin. The halo is specified using a *Halo* parameter in either of these constraint groups. Typical use here would be to allow routing in poly near poly pins, or some other relaxation of standard constraints to be able to route to a pin.

See also Custom Parameter Value (for Process Rule Override Parameters)

- ❑ **Halo**
- ❑ **Top Layer**
- ❑ **Bottom Layer**

Corresponds to a valid layers constraint in the PRE. The valid layers constraint should always contain a contiguous set of layers to use for this net/net group. The router should route using only these layers.

- ❑ **Purposes**

Specify purpose of use, for example (*all*), *high_voltage*, *low_voltage*.

- ❑ **Vias Allowed**

Corresponds to a valid vias constraint in the PRE.

- ❑ **Widths Minimum (for each valid layer)**

minWidth constraints (for each layer matching a valid layer).

- ❑ **Cuts Minimum (for each cut layer)**

The min num cuts constraints (for each cut layer matching a valid via).

- ❑ **(Input Taper) Halo**

- ❑ **(Output Taper) Halo**

Taper halo constraint (in input or output group).

- ❑ **Spacing**

Specify the (Input/Output) taper spacing.

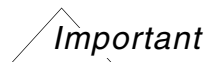
Note: The constraint parameter editor will display *table* for table values.

Note: The above parameters can be used as shortcuts to edit process rules with the PRE. You can edit the most commonly changed parameters for routing constraints directly in the *Constraint Manager*. Furthermore, if there are no overrides in the PRE for these values, a lookup is performed to let you know which values the router will use for routing (if you hover over them, the associated tooltip will indicate what the value for each parameter will be along with a note about where in the lookup that parameter can be found). The constraints created by these parameters will always be in the constraint group above it in the list (or found via lookup).

Members

A single net or design.

Note: No selection is required for a PRO constraint to be created on a design, as it is the default should no other valid member be selected.



A PRO constraint can also be created on a net class, via its default CG parameter, but it will be displayed as *Net Class* constraint, not a separate PRO constraint.

Supporting Applications

- Virtuoso Layout Suite XL (Create Wire Editing)
- Virtuoso Space-based Router
- Modgen Editor

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Process Rule Override Parameters and the Process Rule Editor

The following process rule override constraint parameters are reflected in [The Process Rule Editor](#). That is, changing the value of these parameters in the *Constraint Manager* will make equivalent changes in the PRE and vice versa.

Note: Each parameter tooltip also provides information on where, in the constraint lookup, a value was found.

PRO Parameter	PRE Equivalent Parameter Name	Notes
allowed layers, input taper layer, output taper layer	<i>Valid Layers</i> in default, input taper, and output taper constraint groups respectively.	Specifies the min and max of a range of valid layers that routers should use on a net when tapering. If the database does not currently contain a sequential list of layers then these fields will be blank. Note: In the UI, enter a top and bottom layer entry to the specify the range.
allowed vias	<i>Valid Vias</i> on default constraint group.	Specifies a list of vias that a router should use when routing a net.
layer widths, input widths, output taper widths	<i>Minimum Width</i> for net's default, input taper, and output taper constraint group respectively.	Specifies the minimum width that the router should use for each valid layer. Note: The UI tree display, when unexpanded, will show a summary if there are different values, or a single numeric if the values are all the same value. Changes made to the value here will collectively change all of the values. If you expand the tree you will see the individual layer values.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

via min num cut	<i>Min number of cuts</i> on the net constraint group.	Specifies the minimum number of cuts that a router should use when routing. This parameter is a table-based rule, where the entry in the <i>Constraint Manager</i> supports only the setting/viewing of the first entry in that table. If a table is found with more than one entry, it will not be shown in the <i>Constraint Manager</i> (in this case you should use the PRE to edit or view it). The <i>Constraint Manager</i> will allow the setting of min num cut for each cut layer that participates in at least one valid via. Note: In the UI, enter the number of “counts” (rather than measurements in user units).
input taper halo, output taper halo	<i>Taper Halo Input/Output</i> in taper constraint group.	Specifies a halo within a router that should follow other process rules inside the taper constraint group. Note: In the UI, enter the user unit measurement.

Custom Parameter Value (for Process Rule Override Parameters)

The *Custom* parameter value can be used for a number of Process Rule Override constraint parameters including *Default*, *Input Taper*, and *Output Taper*.

This can be used in cases where a net’s default constraint group contains a mixture of individual constraints and references to one, or more, constraint groups (from design or technology information in the *Constraint Manager*). The *Constraint Manager* will display this scenario by listing what is currently present, separated by “+”.

For example, if the default constraint group of a net contained *2xWidthDoubleCuts* and a further valid via constraint, to additionally restrict vias specifically on this net, then the *Constraint Manager* assistant would display this as “*2xWidthDoubleCuts + Custom*”.

Rail

Definition

The Rail constraint allows you to specify rails in the layout for nets, such as power and ground. It can be specified in the Constraint Manager in either schematic or layout window.

To specify a rail constraint, you just need to select a net in the design. The constraint will be created with the net and the prBoundary as members.

Parameters

In addition to the parameters *Name*, *Enabled*, and *Notes*, you can set the following parameters:

■ Direction

Type: enum

Default: horizontal

Valid Values: horizontal, vertical

On Constraint: Y

On Member: N

Display: Y

User Units: N

■ layerName

Type: integer

Default: 0

Valid Values: layer names

On Constraint: Y

On Member: N

Display: Y

User Units: N

■ Width

Default: 0 [Database Units]

Valid Values: positive

On Constraint: Y

On Member: N

Display: Y

User Units: Y

■ OffsetType

Type: enum

Default: "left" (when vertical Direction selected)

"top" (when horizontal Direction selected)

Valid Values: "left" "right" (when vertical Direction selected) "top" "bottom" (when horizontal Direction selected)

On Constraint: Y

On Member: N

Display: Y

User Units: N

■ Offset:

Type: float

Default: 0 [Database Units]

Valid Values: positive

On Constraint: Y

On Member: N

Display: Y

User Units: Y

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Members

The Rail constraints can be modified at any time either in schematic (and transferred to layout) or directly in layout. Once the modification is done, the constraints show up as failed in the Constraint Manager window.

Supporting Applications

- Virtuoso Analog Auto Placer
- Virtuoso Layout Suite XL (Constraint Aware Editing)

Shielding

Definition

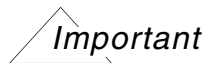
A Shielding constraint enables specialty routing between nets.

The unique net member in the constraint is considered as the “shielded” (or protected) net. That is, it is the net that will be surrounded by a different net. The “different net” that protects the shielded net is considered as the “shield” (or protective) net.

A “shield” net can be used in other Shielding constraints, but only as a shield net, never as a “shielded” net. A “shielded” net can belong only to one Shielding constraint.

When specifying the net width and spacing from the shielded net to the shielding nets, you should use [The Process Rule Editor](#).

Parameters



For additional information on the available shielding parameters see [Shielding Parameters and the Process Rule Editor](#).

■ Shield Net

The *Shield Net* parameter is used to specify the net that is to be considered as the “shield” net.

For more information, see [Creating a Shielding Constraint and Setting a Shield Net](#).

■ Shield Net2

The *Shield Net2* parameter is used to specify the second net that is to be considered as the “shield” net.

■ Shielding

This constraint group contains the following parameters: *msShieldStyle*, *Parallel Widths*, *Parallel Space*, *Tandem Width*, and *Use Existing Shape As Shield*.

The *Shielding* parameter is used to enter process rules related to how nets should be shielded.

❑ **msShieldStyle**

Specifies the shielding style. You have the following choices of styles: *Default*, *Parallel*, *Tandem*, *Coaxial*, *TandemSplit*, *ParallelViaAlign*, *TandemSplitViaAlign*, and *crossTie*.

Note: The `crossTie` shielding style is not supported in Virtuoso Space-based Router and Wire Editor.

The *msShieldStyle* constraint parameter set within the *Constraint Manager* assistant is in the Shielding constraint group of a net, similar to other overrides such as *minWidth*. If no *msShieldStyle* has been specified, *Default* is displayed and is treated as equivalent to *Parallel* by the Virtuoso Space-based Router.

Note: *msShieldStyle* is an EDI-specific parameter.

❑ **Parallel Widths (for each layer)**

Minimum width in the shielding constraint group in the PRE. Specifies the width of the parallel shield on both sides of the signal net in tandem or parallel shield style.

❑ **Parallel Space (for each layer)**

Minimum spacing in shielding constraint group. Specifies the spacing between the shielded net and the shields, on either side, in tandem or parallel shield style.

❑ **Tandem Width (for each layer)**

Tandem width in shielding constraint group. Specifies the width of the shield when it is used above or below the signal net.

❑ **Use Existing Shapes As Shield**

When this parameter is set to *true* and power rails exist adjacent to a shielded net, then these power rails are used as shielding nets. New shielding nets are created if no existing power rails are found.

❑ **Ignore Shielding On Layers**

Note: Available only if the *msShieldStyle* constraint parameter is set to *Parallel*.

Techfile Name: ignoreShieldingOnLayers

Specifies the layers on which shielding should be ignored. When you click the field adjacent to this constraint parameter, the Select Layers form is displayed to enable selection of the specific layers to be ignored. By default, this constraint parameter is set to *none*.

❑ **Style Around Vias**

Note: Available only if the *msShieldStyle* constraint parameter is set to *coaxial*, *parallel*, or *crossTie*.

Techfile Name: styleAroundVia

Determines the shielding style that should be used to add shield wires around the vias. The drop-down list box lets you select one of the following values: *none*, *around*, *allAround*, and *enclose*.

By default, this constraint parameter is set to *enclose*. Selecting this value adds shield wires all around the vias in the preferred direction.

When you select *none*, shield wires are not added around the vias.

around determines whether shield wires should be added around vias.

allAround ensures that shield wires are used all around vias even in the wrong direction, if specified.

❑ Tie Shield

Note: Available only if the *msShieldStyle* constraint parameter is set to *coaxial*, *parallel*, or *crossTie*.

Techfile Name: tieShield

Specifies whether ties should be added for tying the new shield wires to their shield nets. By default, this constraint parameter is set to `true` to ensure that shield ties are added.

❑ Cross Tie Interval (for each layer)

Note: Available only if the *msShieldStyle* constraint parameter is set to *crossTie*.

Techfile Name: msLayerCrossTieInterval

Cross-tie intervals can be specified for individual cross-tie shielding layers associated with a net. This defines a new OA layer constraint, `msLayerCrossTieInterval`.

The *Cross Tie Interval* constraint parameter specifies how often cross-ties should be built on a per-layer basis. The value is an integer that specifies the intervals as a number of tracks between cross-ties (not microns) built on a given layer. For example, a value of 0 (default) results in a cross-tie on all tracks. A value of 1 results in a cross-tie on the alternate tracks. A value of 2 results in a cross-tie after every two tracks.

❑ **Cross Tie Shield Layers**

Note: Available only if the *msShieldStyle* constraint parameter is set to *crossTie*.

Techfile Name: msShieldingLimit

Specifies the shielded layers that should have cross-ties on them. This is done to ensure that specific layers are available for other purposes. The layer pair specified and the layers in between define the continuous set of layers that can be used to build cross-tie shields. For example, if the specified layer pair is *Metal2* and *Metal7*, the cross-tie shields can be built on *Metal2*, *Metal3*, *Metal4*, *Metal5*, *Metal6*, and *Metal7*, but not on *Metal8* or above layers.

Note: The *Top Layer* should be above or same as the *Bottom Layer*.

❑ **Cross Tie Layers**

Note: Available only if the *msShieldStyle* constraint parameter is set to *crossTie*.

Techfile Name: msCrossTieLayers

Setting this constraint parameter indicates that the cross-tie shields should be built for routes on the net on *Top Layer*, *Bottom Layer*, and the layers in between. For example, if the layer pair specified is *Metal3* and *Metal6*, then the cross-tie shields will be built for routes on the net on *Metal3*, *Metal4*, *Metal5*, and *Metal6*.

Note: The *Top Layer* should be above or same as the *Bottom Layer*.

❑ **Tolerance**

■ **hierarchicalScope**

For more information, see The Process Rule Editor and Shielding Parameter Values.

Members

One net only.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Supporting Applications

- Virtuoso Space-based Router
- Innovus System
- Create Wire (supported in pathSeg mode but not path mode)

Note: For Create Wire to work successfully, ensure that Constraint Aware Editing mode is enabled.

Shielding Parameters and the Process Rule Editor

The following shielding constraint parameters allow for editing of process rules in [The Process Rule Editor](#). That is, changing the value of these parameters in the *Constraint Manager* will make equivalent changes in the PRE and vice versa.

Note: Each parameter tooltip also provides information on where, in the constraint lookup, a value was found.

Shielding Parameters	PRE Equivalent Parameter Name	Notes
■ parallel width	<i>Shield Width</i> , <i>Shield Space</i> , <i>Tandem Width</i> , respectively, on the shielding constraint group.	■ <i>Shield Width</i> specifies the parallel shield width to be used if parallel shielding is performed by the router.
■ parallel space		■ <i>Shield Space</i> specifies the parallel shield spacing to be used if parallel shielding is performed by the router.
■ tandem width		■ <i>Tandem Width</i> specifies the tandem shield width to use if tandem shielding is to be performed by the router.

Note: All are set on a per layer basis.

Creating a Shielding Constraint and Setting a Shield Net

To create a Shielding constraint and set the associated shielding net:

1. Select the net that is to be shielded.

Note: If you select multiple nets, and try to apply a *Shielding* constraint, a consistency check warning message is displayed in the CIW.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

This selection can be made using the *Navigator* assistant, *Search* results, or directly from the design canvas.

2. Select the *Shielding* (*Routing* drop-down menu) constraint from the Constraint Generator menu on the *Constraint Manager*.

This will create a *Shielding* constraint for the selected net.

3. Select the newly created *Shielding* constraint in the *Constraint Manager*.

Note: The *Shielding* constraint will have one member, the net you selected in Step 1.

4. Using the *Shield Net* and *Shield Net2* parameters, in the *Constraint Parameter Editor* section of the *Constraint Manager*, select the net that is to be used as a shielding net by either:

- a. Choosing the net from the drop-down list box for the *Shield Net* parameter.

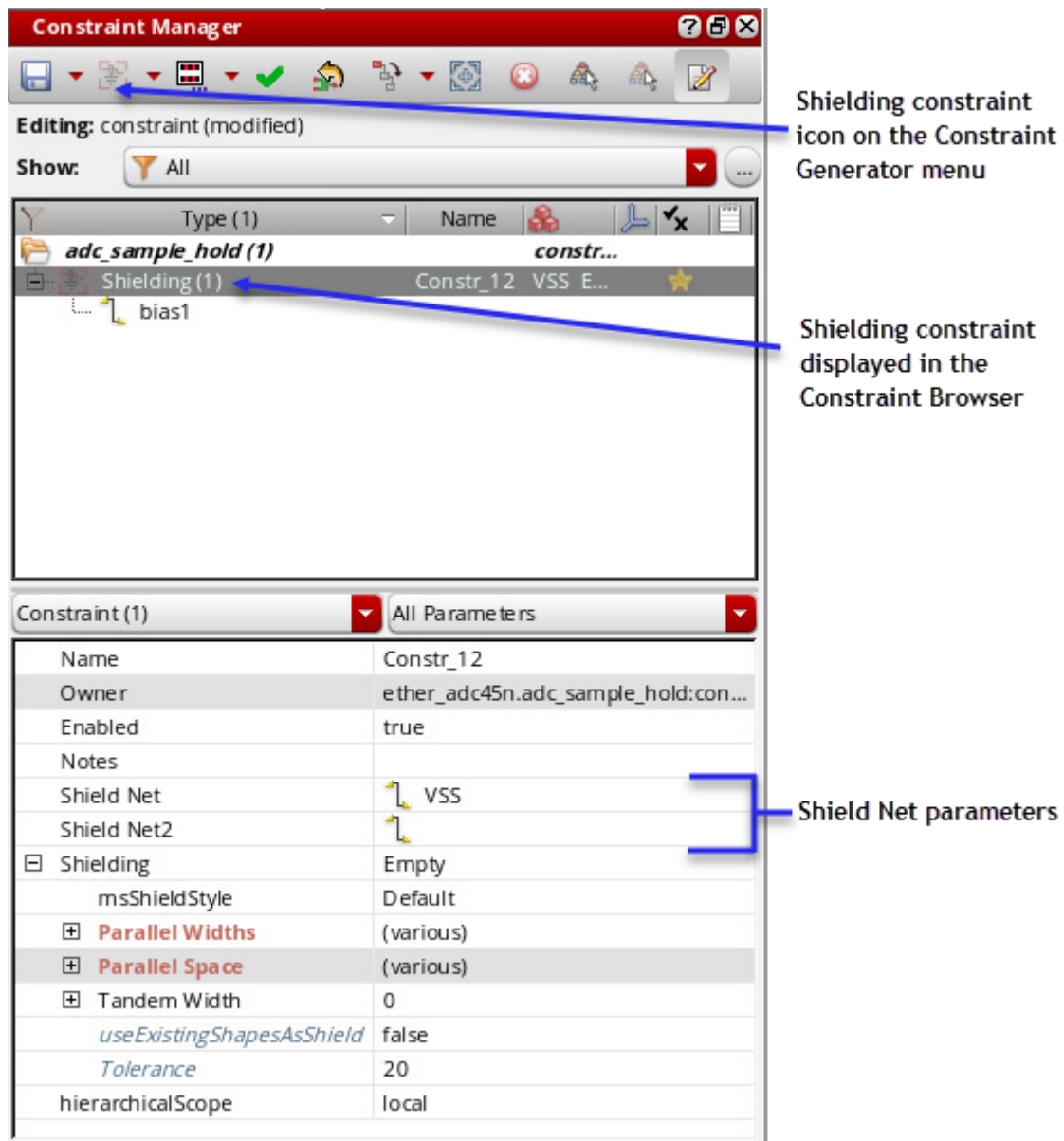
You will be displayed a list of all the nets which have been registered with the *Circuit Prospector* as supply nets or have their sigType set to power or ground.

- b. Entering the shielding net name manually into the *Shield Net* parameter entry field.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

- c. Clicking the white arrow icon to the right of the *Shield Net* parameter which will allow you to interactively select, using the mouse, a shielding net from the *Navigator*, *Search* toolbar/assistant results, or directly from the design canvas.



Note: When in interactive selection mode, you can still use other commands such as copy, move, or zoom. Press the **ESC** key to exit interactive selection.

Shielding Parameter Values

Note: See also [The Process Rule Editor](#)

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

The following values can be set for the *Shielding* parameter:

■ *Empty*

The default value applied when a *Shielding* constraint is first created. The rule group (*shielding*) is created and attached to the selected net. There are no additional process rules associated with the shielding constraints.

■ *Custom*

When selected, the PRE is invoked by default displaying the selected objects.

- ☐ If only the *Shielding* constraint is selected (in the *Constraint Manager*), then the currently selected object is shown in the PRE's constraint tree. That is, the *Shielding* constraint and its process rules are displayed.
- ☐ If the selection also includes the net to which the constraint was applied, then the PRE will display the *Shielding* constraint and any other constraint in that net.

■ *Parallel (Generate)*

Selecting this value will auto-generate a process rule for the *Shielding* constraint that corresponds to parallel shielding. These rules are shown in the PRE as user-defined constraint groups, populated by *shieldWidth* (the width of the possible metals) and *shieldGap* (the gap between the shielding net and shielded net) constraints. The rules are auto-generated, but can be considered as a template that can be modified as required, either by directly editing the generated constraint group on the design tab of the PRE, or copying and pasting the rules to build the desired constraints.

■ *Tandem (Generate)*

As above, but this time a *Tandem* shielding constraint will be generated, with *Tandem Layer Above*, *Tandem Layer Below*, and *Tandem Width* constraints on the allowed layers. Again, this can be considered as a template for any required customizations.

■ *Coaxial (Generate)*

Again, as above, but this time a *Coaxial* shielding constraint is generated. The constraint template created here contains combined *Tandem* and *Shield* rules (a combination of *Parallel* and *Tandem*).

Note: When one of these “generator” values has been selected, the associated option in the PRE pull-down menu will change from *generate* to *design*. This indicates that the associated process rule has been created and it is now in the design (for example, *Coaxial (Generate)* will change to *Coaxial (Design)*).

Symmetry

Definition

The Symmetry constraint is used to define symmetries of instances, nets, modgens and pins about an axis. It allows mirrored and non-mirrored symmetries which can be specified via the constraint's parameter mirror. The Symmetry constraint accepts an axis. If no axis is specified, a default axis that has default parameters is considered.

When you select a symmetry constraint in the Constraint Manager, the members as well as the associated axis are also highlighted (haloed) in the Layout graphical area.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Parameters

In addition to the parameters *Name*, *Enabled*, and *Notes*, you can set the following parameters:

- **Axis** (see [Axis Parameter](#))

- **Mirror**

Type: Boolean

Default: true

Valid Values: true/false

On Constraint: Y

On Member: N

Display: Y

User Units: N

The mirror parameter is only used on objects that have an orientation, such as instances. For example, as pins are shapes, and do not therefore have an orientation, the mirror parameter would be ignored in their case.

- **scope**

Type: enumSet

Default: boundary

Valid Values: “boundary”, “contents”

On Constraint: Y

On Member: N

Display: Y

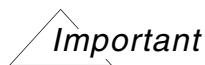
User Units: N

■ Check With Halo

This parameter applies to instance and net members. The value of this parameter defines the type of symmetry check that needs to be applied to the constraint:

- ☐ `nil`: Simple check without halo
- ☐ `boundary`: Check with halo based on cell boundary
- ☐ `per-layer`: Check with per-layer halo

Note: For net symmetry only `nil` and `boundary` values are valid. In this case `boundary` should be interpreted as check with halo, which must be applied to the constraint with net members.



The constraint groups discussed below are also applicable to net symmetry constraints.

■ hierarchicalScope

Note: This parameter is visible only when the Symmetry constraint includes net members.

Specifies the current hierarchical scope of the constraint: *local*, *below*, or *above* the current cell level.

■ Default Group (Constraint Group)

Note: This parameter is visible only when the Symmetry constraint includes net members.

The *Default Group* category contains the following parameters: *Top Layer*, *Bottom Layer*, *Purposes*, *Vias Allowed*, *Widths Minimum*, *Cuts Minimum*, and *Spacing*. These parameters are used to reflect the contents of a Symmetry (including Net Symmetry) default constraint group.

Constraints can be placed in the default constraint group to locally override foundry rules obtained through standard lookup. Adding a constraint parameter to a Symmetry default constraint group is equivalent to adding the same constraint in each net member's default constraint group. If there are similar constraints in the net's default constraint group, and the Symmetry default constraint group, then the net's default constraint group parameter settings will have precedence.

A typical use here would be to route a set of nets with a specific wider width or a net with a specific number of via cuts, or to modify the valid layers for all nets in a symmetry constraint.

In addition to the parameters listed above, the *Default Group* category has the *Wire Types* parameter that can be used to define a set of wire types for each layer. For more information, see [Assigning a Wire Type to a Net](#).

For more information on the Default Group, see [The Process Rule Editor](#).

For more information on customizing constraint groups, see [Customizing a Constraint Group](#).

■ Within Group (Constraint Group)

Note: This category is visible only when the Symmetry constraint contains nets as its members (that is, not a nested net class).

The parameters available in the *Within Group* category help to define the process rules for the net members in a Symmetry constraint. These parameters include [Spacing](#) and [Tolerance](#). For more information on the *Within Group* parameters, see [The Process Rule Editor](#).

- The *Tolerance* parameter describes the percentage allowed deviation from specification when implementing a given constraint. This parameter applies only to symmetry (and other constraints) on nets, for the purpose of routing.

- ❑ In addition to the *Spacing* and *Tolerance* parameters, you can define the *Same Mask* parameter to ensure all shapes on a given layer for a net are on the same mask. This parameter allows you to specify the color of the nets in the net class. This can also have a hard or a soft attribute.

Note: The *Same Mask* parameter is equivalent to using the `msSameMask` constraint. If you reset the value of this constraint, the color of the nets should be reset accordingly.

- ❑ Optionally, in addition to the *Same Mask* parameter, you can set the *Mask Name* parameter to ensure that the shapes will be on the specified color mask for each routing layer for the nets in the Symmetry constraint.

For more information, refer to [Specifying the Color Mask for Critical Nets in Schematic](#).

■ **Group to Outside Group (Constraint Group)**

Note: This parameter is visible only when the Symmetry constraint includes net members.

It contains the following parameters: *Spacing*, *Tolerance*, and *hierarchicalScope*.

The *Group to Outside Group* parameters are used to specify spacing for the routed symmetry constraint to everything else in the design. These parameters can be used to enter process rules between nets in a Symmetry constraint and those nets that are not part of that constraint. For more information on the *Group to Outside Group* parameters, see [The Process Rule Editor](#).

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Members

One (self-symmetry) or two objects of the same object type. The object type can be: instance, net, modgen constraint (selected in the *Constraint Manager*), modgen figGroup, regular figGroup, terminal, shape, or pin.

- Any self-symmetrical object can be in two orthogonal constraints.
- Any object in a symmetrical pair can be in four symmetrical constraints (two vertical and two horizontal) as long as the two vertical and horizontal axes are different.

Note: For sharing object members, the constraint must have explicit symmetry axes and not the default Axis Parameter.

Member order is important. When multiple pair or objects, for example, (A B) (C D) and (net1 net2), share the same symmetry axis, the object order may be important to analog placers and routers. If the order of the objects is (A B) (C D), that implies that A and C are on the same side of the symmetry axis and B and D on the opposite side.

Note: A symmetry (net) constraint can also be a *constraint container*. That is, it can contain its own members and also be used as a member itself for other constraints.

Supporting Applications

- Virtuoso Custom Digital Placer (*all*)
- Virtuoso Space-based Router (*net symmetry only*)
- Virtuoso Analog Auto Placer (*not shape symmetry*)
- Virtuoso Layout Suite XL / Constraint Aware Editing (*not shape symmetry*)

Voltage Synced Net

Definition The Voltage Synced Net constraint identifies whether a pair of synced nets transition in phase with each other, and their delta voltage is calculated differently to the default `minVoltageSpacing` calculation. The calculation for synced nets is `deltaV =`

$$\text{Max}(\text{abs}(\text{net1\_maxV} - \text{net2\_maxV}), \text{abs}(\text{net1\_minV} - \text{net2\_minV}))$$

The constraint acts as a placeholder for DRD (Design Rule Driven Editing) by identifying the synced nets that require their delta voltage checked. DRD generates spacing markers for any shapes on synced nets that fail `minVoltageSpacing` based on this calculation of `deltaV`.

Additionally, it is not possible to have multiple voltage synced net constraints on the same net.

(IC23.1 EXL and Higher Tiers) Virtuoso's VDR flow generates voltage synced shapes on the appropriate LPP for each pair of nets in a group that has this Voltage Synced Net constraint. For more details see [Defining and Checking Voltage Synced Nets](#).

Note: The [voltageLayerMarkerMapping](#) and [voltageLayerPairMarkerMapping](#) constraints specify the LPPs on which the voltage synced nets are generated.

Parameters

In addition to the parameters *Name*, *Enabled*, and *Notes*, you can set the following parameter:

■ hierarchicalScope

Specifies the current hierarchical scope of the constraint as *local*, *below*, or *above* the current cell level.

For more details see [Pushing and Pulling Constraints in a Schematic or Layout Hierarchy](#).

Members

Two or more nets.

Supporting Applications

- Virtuoso Design Rules
- Virtuoso Voltage Dependent Rules Flow

Note: The fixed constraint cannot be stored as a template as it is an Open Access object

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

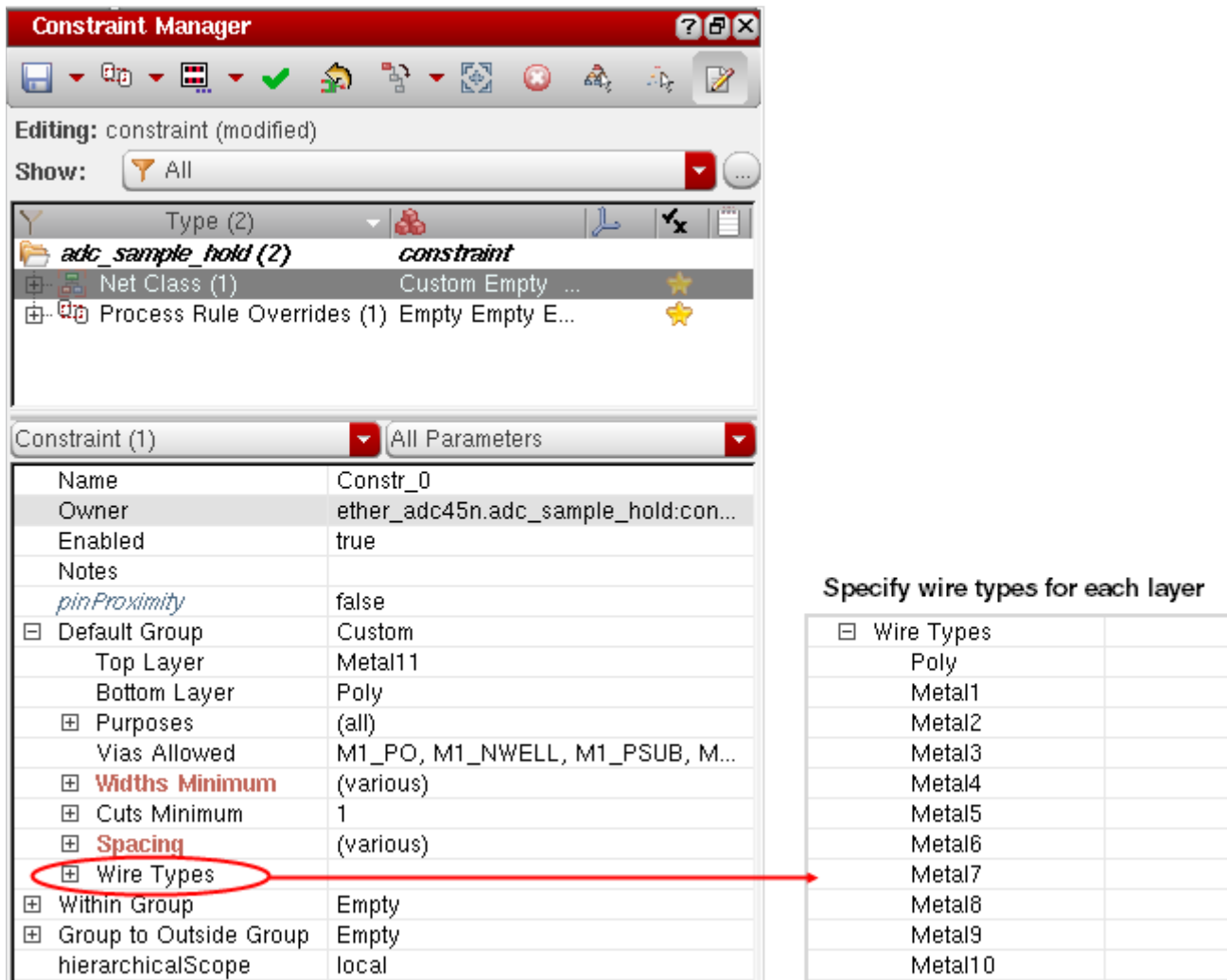
attribute and does not have independent OA storage. For more information, see [Constraint Template Storage](#) in the Virtuoso Unified Custom Constraints Configuration Guide.

Delta Voltage

Definition	Specifies the delta voltage between two nets. To create a delta voltage constraint, you need to specify a valid value, which should not be a negative value. For more information, see Creating Delta Voltage Constraints between Nets. The constraint defines the maximum voltage difference that can exist between two nets. Any value greater than the defined value results into constraint violation. Delta Voltage option is greyed out when only a single net or more than two nets are selected. A warning displays in CIW if one net (out of the two) is deselected from the deltaVoltage constraint. Same is the case if a net is added in an already existing deltaVoltage constraint.
Parameters	In addition to the parameters <i>Name</i>, <i>Enabled</i>, and <i>Notes</i>, you can set the following parameter: ■ deltaVoltage Specifies the current delta voltage of the constraint.
Members	Two nets only.
Supporting Applications	■ Virtuoso Design Rules ■ Virtuoso Voltage Dependent Rules Flow

Assigning a Wire Type to a Net

In the Constraint Manager assistant, the following constraint types allow you to assign a specific set of wire types for each layer: Bus, Diff Pair, Matched Length, Net Class, Process Rule Overrides, and Symmetry. To do this, use the *Wire Types* parameter listed in the Constraint Parameter Editor pane under the *Default Group* category, as shown in the figure below.



The screenshot shows the 'Constraint Manager' window with the 'Editing: constraint (modified)' tab. The 'Show:' dropdown is set to 'All'. The 'Type (2)' dropdown is set to 'constraint'. The 'Constraint (1)' dropdown is set to 'All Parameters'. The 'Wire Types' parameter is highlighted with a red circle. A red arrow points from the 'Wire Types' parameter to the table below.

Specify wire types for each layer	
☐ Wire Types	
Poly	
Metal1	
Metal2	
Metal3	
Metal4	
Metal5	
Metal6	
Metal7	
Metal8	
Metal9	
Metal10	

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

The wire types should be specified as a comma-separated list of strings, as shown in the figure below.

<i>pinProximity</i>	false
[-] Default Group	Custom
Top Layer	Metal11
Bottom Layer	Poly
[+] Purposes	(all)
Vias Allowed	M1_PO, M1_NWELL, M1_PSUB, ...
[+] Widths Minimum	(various)
[+] Cuts Minimum	1
[+] Spacing	(various)
[-] Wire Types	(various)
Poly	2X,3X
Metal1	1X,2X
Metal2	3X,4X
Metal3	2X,3X
Metal4	2X,4X
Metal5	3X,4X
Metal6	
Metal7	
Metal8	
Metal9	
Metal10	
Metal11	
[+] Within Group	Empty

Comma-separated wire types for each layer

Blanks for layers without a corresponding entry in technology file

If a comma appears in the wire type name, add a backslash just before the comma to create an escape sequence. For example: `myWire\,1and2,myWire3`.

If a layer does not have a corresponding entry in the technology file, its value is shown as blank in the Constraint Parameter Editor pane. See *Metal6* to *Metal11* in the figure above. You can manually add the required wire types for such layers.

If the wire types assigned to each layer are different, (*various*) is displayed in the cell adjacent to the *Wire Types* parameter.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

To modify the values for all layers simultaneously, enter the required values in the cell adjacent to the *Wire Types* parameter.

Left Screenshot (Initial State)		Right Screenshot (Modified State)	
<i>pinProximity</i>	false	<i>pinProximity</i>	false
Default Group	Custom	Default Group	Custom
Top Layer	Metal11	Top Layer	Metal11
Bottom Layer	Poly	Bottom Layer	Poly
Purposes	(all)	Purposes	(all)
Vias Allowed	M1_PO, M1_NWELL, M1_PSUB, ...	Vias Allowed	M1_PO, M1_NWELL, M1_PSUB, ...
Widths Minimum	(various)	Widths Minimum	(various)
Cuts Minimum	1	Cuts Minimum	1
Spacing	(various)	Spacing	(various)
Wire Types	(various)	Wire Types	1X,2X,3X
Poly	2X,3X	Poly	1X,2X,3X
Metal1	1X,2X	Metal1	1X,2X,3X
Metal2	3X,4X	Metal2	1X,2X,3X
Metal3	2X,3X	Metal3	1X,2X,3X
Metal4	2X,4X	Metal4	1X,2X,3X
Metal5	3X,4X	Metal5	1X,2X,3X
Metal6		Metal6	1X,2X,3X
Metal7		Metal7	1X,2X,3X
Metal8		Metal8	1X,2X,3X
Metal9		Metal9	1X,2X,3X
Metal10		Metal10	1X,2X,3X
Metal11		Metal11	1X,2X,3X
Within Group	Empty	Within Group	Empty

When you view the details of a constraint in the CIW, the *Wire Types* parameter is identified as *allowedWireTypes*. It is of string type and is displayed in the following format:

```
"rowNumber1=list of values:rowNumber2=list of values..."
```

Here, any comma or colon in the name of a wire type will be displayed in an escape sequence.

For example, if you want to view the details of constraint `Constr_0`, run the following commands in the CIW:

```
constraint=ciConFind(ciGetCellView() "Constr_0")
constraint~>parameters
```

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

The following result is displayed:

```

Virtuoso® ICADV12.2-64b - Log: /tmp/CDNS_abha.log
File Tools Options Help
cadence

("default" string "Custom")
("allowedLayers" string "10:30:34:38:42:46:50:54:58:62:66:70")
("allowedVias" string
"M1_P0:M1_NWELL:M1_PSUB:M1_NIMP:M1_PIMP:M1_DIFF:M2_M1:M3_M2:M4_M3:M5_M4:M6_M5:M7_M6:M8_
M7:M9_M8:M10_M9:M11_M10")
("layerWidths" string
"10=90:30=120:34=160:38=160:42=160:46=160:50=160:54=160:58=160:62=160:66=440:70=440")
("viaNumCuts" string "20=1:32=1:36=1:40=1:44=1:48=1:52=1:56=1:60=1:64=1:68=1")
("defSpacing" string "10=120:30=Table Value:34=Table Value:38=Table Value:42=Table
Value:46=Table Value:50=Table Value:54=Table Value:58=Table Value:62=Table
Value:66=Table Value:70=Table Value")
("allowedWireTypes" string
"10=1X, 2X:30=1X, 2X:34=1X, 2X, 3X:38=1X, 2X:42=1X, 3X:46=2X, 3X:50=1X, 2X:54=1X, 2X:58=2X, 3X:62
=1X, 2X:66=1X, 2X:70=1X, 2X")
("reflexive" string "Empty")
("refSpacing" string "10=120:30=Table Value:34=Table Value:38=Table Value:42=Table
Value:46=Table Value:50=Table Value:54=Table Value:58=Table Value:62=Table
Value:66=Table Value:70=Table Value")
("msTolerance" float 20.0)
("msSameMask" boolean 0)
.. .. ..

constraint=ciConFind(ciGetCellView() "Constr_0")
constraint->parameters
|

mouse L: schSingleSelectPt() M: constraint->parameters R: schHiMousePopUp()
1 >

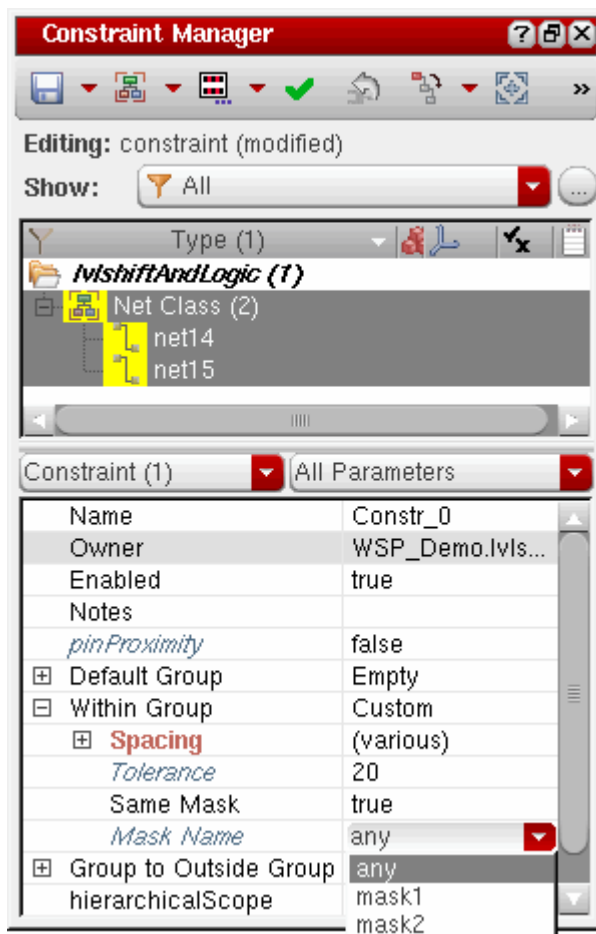
```

Specifying the Color Mask for Critical Nets in Schematic

Critical nets can be identified in the schematic and then routed on a single mask for each layer. In addition, you can optionally specify the color mask.

To specify color-specific constraints, in the Constraint Manager assistant accessed from Schematic XL, the following parameters are available for setting in the *Within Group* category for the Diff Pair, Net Class, and Symmetry constraints that contain critical nets:

- *Spacing* parameter and/or pre-coloring spacing constraint groups.
- *Same Mask* parameter set to `true` specifies that all shapes for the nets in the net class must be on the same mask for each routing layer.
- (Optional) *Mask Name* parameter, as shown below, ensures that shapes will be on the specified color mask for each routing layer for the nets in the net class.



Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

The *Mask Name* can be set to the desired mask using the drop-down list box that provides the following options:

Selection	Description
any	Shapes on the nets will be on the same color mask for each routing layer. This is the default.
mask1	Shapes for the nets will be on <code>mask1</code> for each routing layer.
mask2	Shapes for the nets will be on <code>mask2</code> for each routing layer.

Important

The choices displayed in the *Mask Name* drop-down list box are dependent on the number of masks specified in the `techLayers` section of the technology file. For example, if the number of masks is 1, `mask2` will not be displayed for choice.

To update constraints in the layout to match those set in the schematic, do one of the following in the layout view:

- Choose *Connectivity – Update – Layout Constraints* on the menu bar.
- Click the *Update Layout Constraints* button on the Constraint Manager toolbar.

Using colorMask Constraint Parameter of msSameMask

Add the `colorMask` parameter to the `msSameMask` constraint definition to specify the color mask. This parameter is represented by an integer value as shown in the table below:

Value	Selection	Description
0	any	Shapes on the nets will be on the same color mask for each routing layer. This is the default.
1	mask1	Shapes for the nets will be on <code>mask1</code> for each routing layer.
2	mask2	Shapes for the nets will be on <code>mask2</code> for each routing layer.

Constraints in Virtuoso Parasitic Aware Design Environment

The following constraints are related to Virtuoso Parasitic Aware Design:

- [Parasitic Estimate](#)
- [Parasitic Filter](#)

Parasitic Estimate



The Parasitic Estimate constraint is not accessible from the [Constraint Generator](#) drop-down menu. For more information on the use and application of parasitic estimates, see the [Parasitic Estimates Assistant](#) in the *Virtuoso Parasitic Aware Design User Guide*.

Definition The Parasitic Estimate constraint is used to specify R/C parasitic estimates to be met by the layout software. Backannotation will issue warnings when parasitics are not met.

Estimates are in the form of a star-shaped model for each (flattened) net. A resistance can be associated with each leaf-level instance terminal. Capacitance is connected between the central nodes of nets. Each constraint specifies one resistance or capacitance estimate and the parameters R and C are mutually exclusive. Consequently, a number of constraints may be needed to specify a complete model for a net.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Parameters

Note: In addition to the parameters listed below History and Notes information can also be set.

■ R

Type: float

Valid Values: positive

On Constraint: Y

On Member: N

Display: Y

User Units: N

■ C

Type: float

Valid Values: positive

On Constraint: Y

On Member: N

Display: Y

User Units: N

Members

For resistance estimates, one net or two or more terminals.

For capacitance estimates two members are required, both nets. This expresses a coupled capacitance estimate if both are signal nets.

Supporting Applications

- Virtuoso Parasitic Aware Design

Parasitic Filter



The Parasitic Filter constraint is not accessible from the Constraint Generator drop-down menu. For more information on the use and application of parasitic filters, see the Parasitic Filters Assistant in the *Virtuoso Parasitic Aware Design User Guide*.

Definition

The Parasitic Filter constraint allows reduction of parasitic data in extracted views by filtering out parasitic instances. Its members are nets, instances or cell views. Filters always apply to parasitic instances attached to nets. The use of an instance or cellview master is a short-cut for specifying nets contained by that instance or cellview.

The parameter *isCoupled* specifies whether the filter applies to any parasitic attached to the net, or only to coupling parasitics between two specific nets (in the latter case, two members are required).

Parameter *type* defines which type of parasitic should be filtered, for example, one of “R”, “coupledC”, “decoupledC”, “L” and “K”. Multiple filters, each of a different *type*, can be applied to the same members.

Using parameter *include* it is possible to define that:

- *none* of the parasitics should be included.
- *all* parasitics should be included.
- all parasitics over a certain *threshold* should be included.

In the last case listed above, the parameter *threshold* needs to be defined to give the threshold for the parasitics to be included.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Parameters

■ **isCoupled**

Type: bool

Valid Values: false

Valid Values: “true”, “false”

On Constraint: Y

On Member: N

Display: Y

User Units: N

■ **type**

Type: enum

Valid Values: R

Valid Values: “R”, “coupledC”, “decoupledC”, “L”, “K”

On Constraint: Y

On Member: N

Display: Y

User Units: N

■ **include**

Type: enum

Valid Values: threshold

Valid Values: “none”, “all”, “threshold”

On Constraint: Y

On Member: N

Display: Y

User Units: N

■ **threshold**

Type: float

Valid Values: 0.0

Valid Values: positive

On Constraint: Y

On Member: N

Display: Y

User Units: N

Note: Legal only when include = threshold.

Members

Two members if isCoupled is true, otherwise one member. Each member is either a net, instance, or master.

- If the member is a net, the filter applies to all parasitic instances on the net at all levels of hierarchy.
- If the member is an instance, the filter applies to all parasitic instances on all nets that are contained by the instance (i.e. do not escape above it in the design hierarchy).
- If the member is a master, the filter applies to all parasitic instances on all nets contained by all occurrences of the cell view.

**Supporting
Applications**

- Virtuoso Parasitic Aware Design

Axis Parameter

An axis is a reference line that is used by some constraints to better describe the relationship of the constraint members. There can be any number of axes in a design, and multiple constraints may reference the same axis. Constraints that use an axis, such as symmetry, alignment, and distance, have an expandable *Axis* parameter which specifies the axis name.

When expanded, the *Axis* parameter displays the following sub-parameters:

■ *direction*

- ☐ Type = enumSet
- ☐ Default = constraint dependent
- ☐ Valid Values = “horizontal”, “vertical”

■ *axisLocation*

- ☐ Type = enumSet
- ☐ Default = “any”
- ☐ Valid Values = “any”, “fixed”

The value “any” implies that the axis may be moved by editing tools. The value “fixed” implies that the axis is stationary.

■ *coordinate*

- ☐ Type = float
- ☐ Default = 0.0 (user units).

Note: There is only one coordinate as the horizontal axis is `y=Val` for the line.

- ☐ Valid Values = any float

The *coordinate* parameter is read-only when *axisLocation* is set to “any”. If *axisLocation* is set to “fixed”, you may specify the position of the axis with the *coordinate* parameter.

■ *realizedCoordinate*

- ☐ Type = float
- ☐ Default = 0.0 (user units).
- ☐ Valid Values = any float

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Modifying axis parameters will modify the axis. As more than one constraint may reference the same axis, modifying axis parameters may affect multiple constraints.

Note: When transferring axis information, for example on an alignment constraint from schematic to layout, all sub-parameter values will be reset apart from *realizedCoordinate* which will remain unchanged.

Widths Minimum Parameter

The *Widths Minimum* parameter in the [Constraint Parameter Editor](#) defines a particular set of width values for a given layer-purpose pair (LPP) instead of defining a minimum and maximum width. A width is defined as the shorter dimension of a rectangle.

If the technology database contains width definitions, each LPP listed below the *Widths Minimum* parameter displays the corresponding minimum width value in the cell adjacent. You can make the required modifications to these width values as detailed in [Overriding the Widths Parameter Values](#).

When you place the pointer on any LPP name listed below the Widths Minimum parameter, the corresponding width values are displayed in the tooltip. If multiple width values are defined, they are all displayed.

Overriding the Widths Parameter Values

To override a single width value, click the cell adjacent to the corresponding net LPP name and enter the required *Widths Minimum* parameter value.

Multiple width values from a defined range of valid widths associated to a net LPP can be selected for net type constraints: *Bus*, *Diff Pair*, *Matched Length*, *Net Class*, *Process Rule Overrides*, and *Symmetry*. It requires the constraint [allowedWidthRanges](#) to have been defined for the layers.

To override a multiple width value, click the cell adjacent to the corresponding net LPP name to display an ellipsis  button. Click the button to open the [Allowed Width Ranges Form](#) where you can select and deselect multiple widths from a defined range of valid widths associated to a net LPP.

Note: Although a range of widths may be selected, only the smallest width will be displayed on the *Constraint Parameter Editor*.

Allowed Width Ranges Form

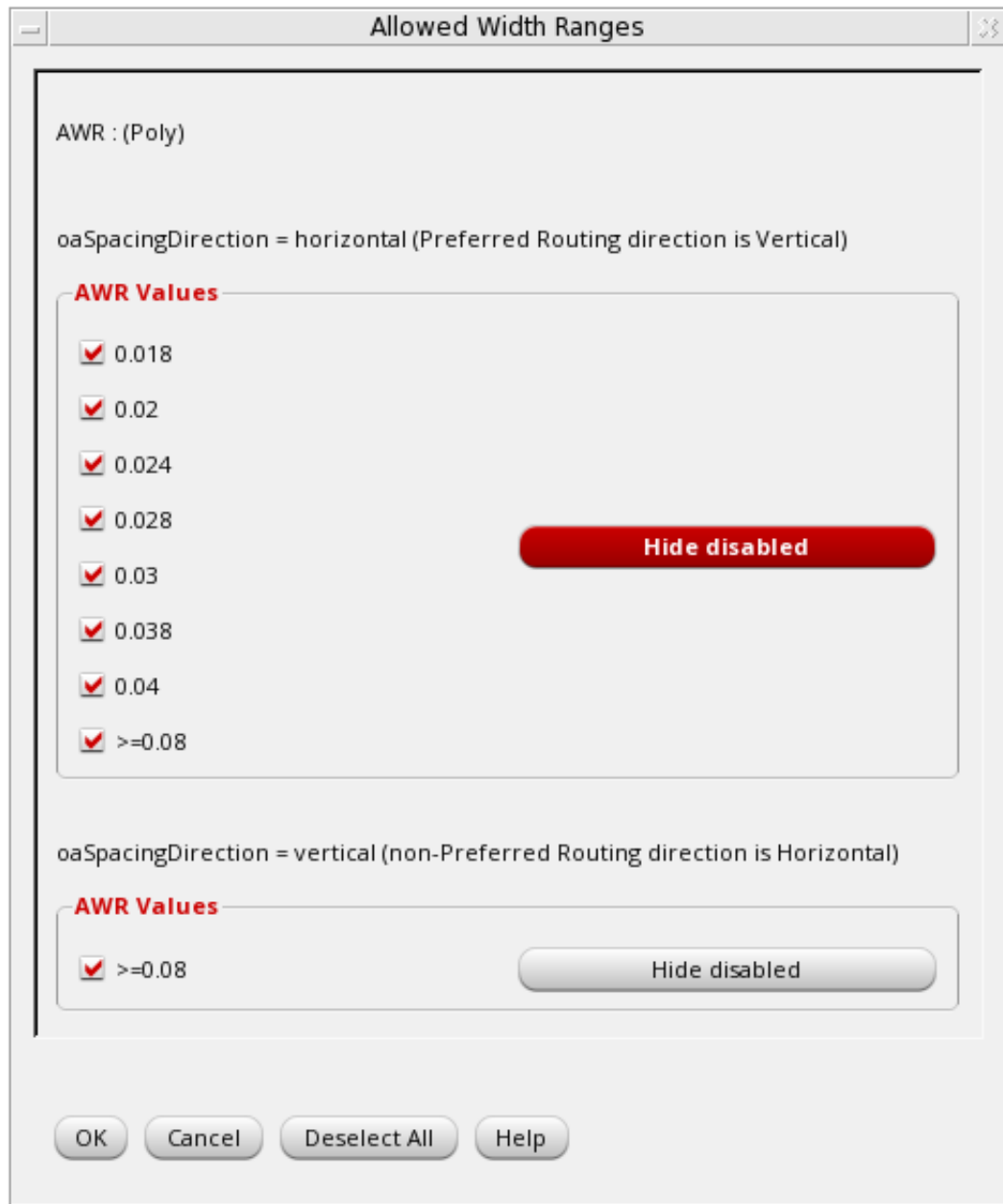
Note: The constraint [allowedWidthRanges](#) needs to be defined for each net LPP in order for this form to be enabled.

This form lets you select and deselect multiple widths from a defined range of valid widths associated with a net LPP. It is accessed from the *Constraint Parameter Editor* by clicking

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

the cell adjacent to a LPP under the Widths Minimum Parameter and clicking the ellipsis  button that is displayed.



The range is displayed, grouped by the direction in which it is measured; horizontal or vertical (if specified). By default, the horizontal range is displayed followed by the vertical range.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

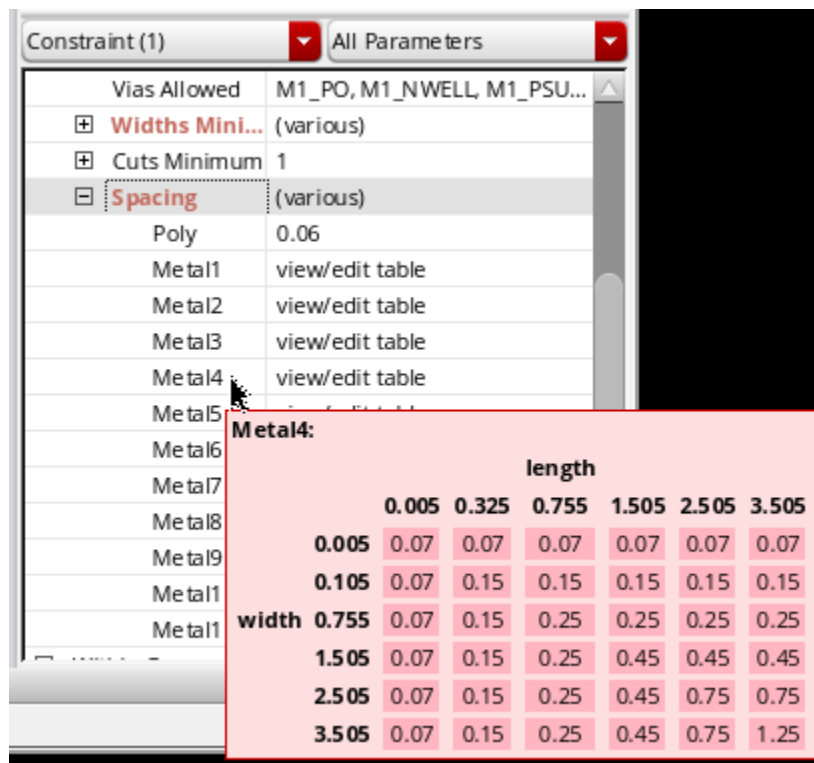
<i>oaSpacingDirection</i>	If specified, this is the direction in which the width of the rectangle is measured. This can be <code>Vertical</code> or <code>Horizontal</code> . If a horizontal and a vertical width range have been set, they are both displayed.
<i>AWR Values</i>	You can select and deselect the widths required from a defined range of valid widths associated for the net LPP.
<i>Show/Hide disabled</i>	Toggles the display of the selected widths.
<i>Select All</i>	Selects all widths in the range.
<i>Deselect All</i>	De-selects all widths in the range.

Spacing Parameter

The *Spacing* parameter in the [Constraint Parameter Editor](#) defines the minimum spacing between a pin and other member pins. It is defined for each layer-purpose pair (LPP) in user units.

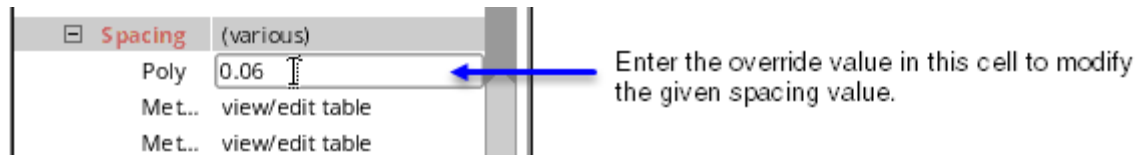
If the technology database contains spacing definitions, the cell adjacent to each LPP listed below the *Spacing* parameter displays the corresponding spacing values. The display of spacing values depends on how they have been defined in the technology database, that is, as a single value or as a one- or two-dimensional table. If an LPP has a single spacing value, it is displayed in the adjacent cell; otherwise, the text *view/edit table* is displayed. You can make the required modifications to these spacing values as detailed in the [Overriding the Spacing Parameter Values](#) section.

When you place the pointer on any LPP name listed below the *Spacing* parameter, the corresponding spacing values are displayed in the tooltip. The figure below illustrates the display of a tooltip with a two-dimensional table of spacing values for *Metal4*.

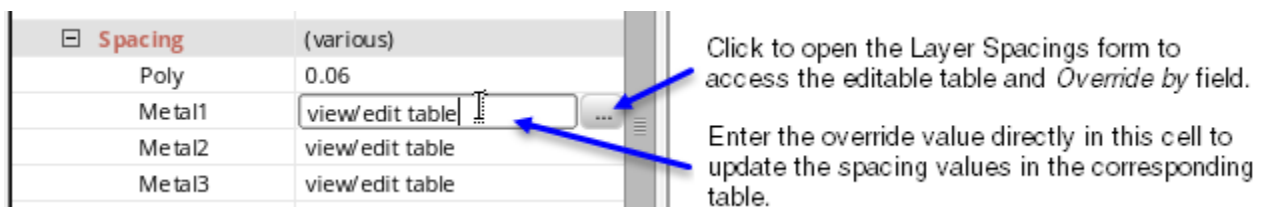


Overriding the Spacing Parameter Values

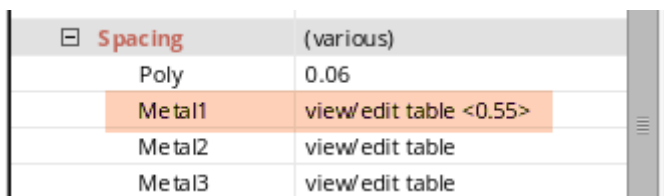
To override a single spacing value, click the cell adjacent to the corresponding LPP name and enter the required *Spacing* parameter value, as shown below.



If the LPP's spacing values are defined in a one- or two-dimensional table, clicking the adjacent cell displays the *view/edit table* field and the  button, as shown below. These two widgets provide you the flexibility to enter a single override value in the displayed field or to view and edit the spacing table in the Layer Spacings form opened by clicking the button.



When you enter a single override value in the *view/edit table* field, it modifies the table values that are less than the entered value and the rest of the values remain unchanged. In addition, the override value starts to show within angular brackets corresponding to *view/edit table* in the Constraint Parameter Editor, as highlighted below, and the tooltip display gets updated accordingly.

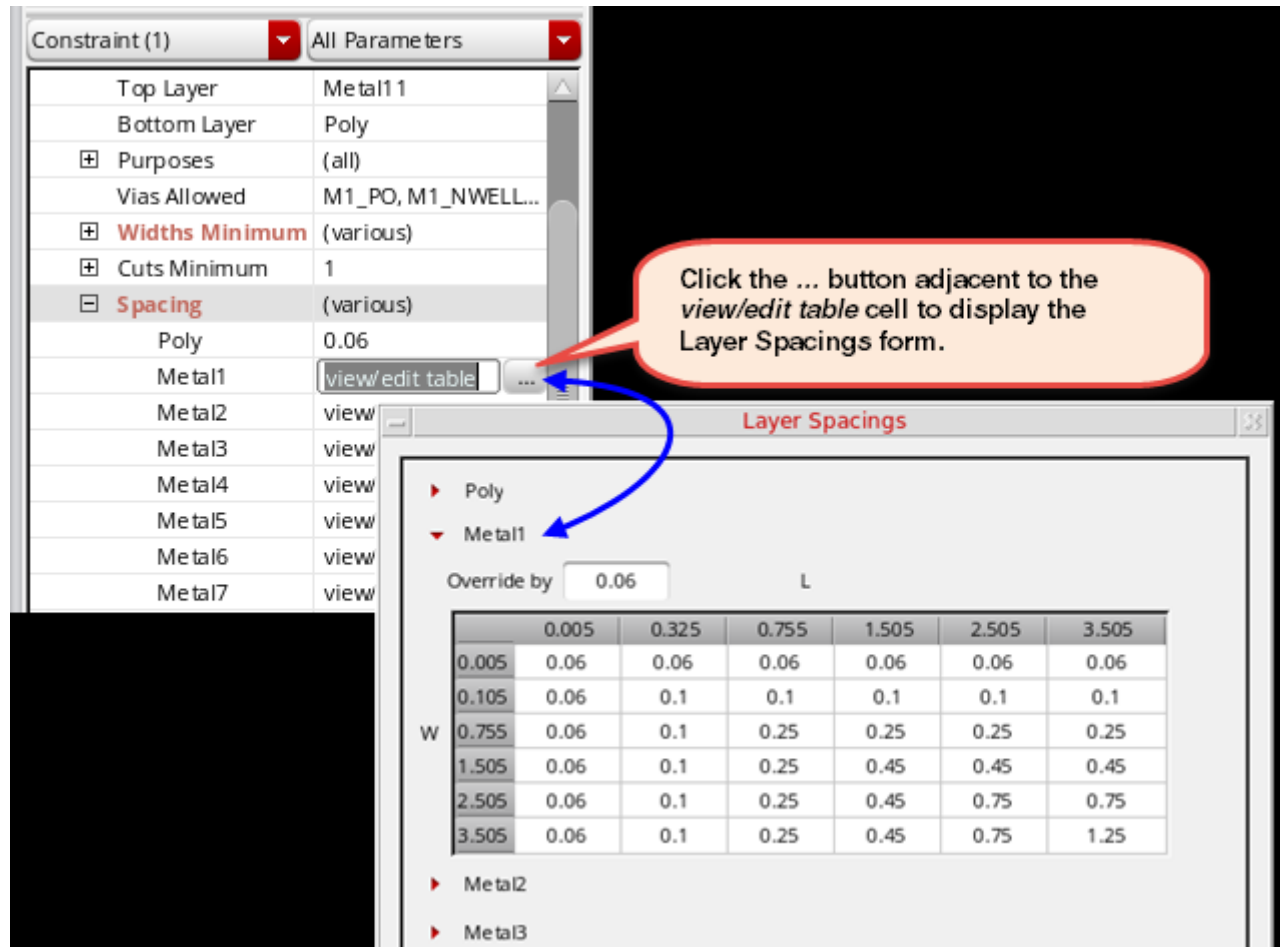


When you access the Layer Spacings form, the section corresponding to the selected LPP expands by default and contains the spacing table for that LPP. The sections containing details corresponding to the remaining LPPs appear as collapsed. For example, when you click the  button adjacent to the *view/edit table* field for *Metal1*, the Layer Spacings form

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

opens with the *Metal1* section expanded to show the relevant spacing table. The other sections such as *Poly*, *Metal2*, *Metal3*, and so on are collapsed, as shown below.



Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

You can expand the other LPP-specific sections to view and edit the related spacing values, as shown below.

The screenshot shows the "Layer Spacings" dialog box. It has a title bar with the text "Layer Spacings". Inside, there are sections for different metal layers. The "Metal1" section is expanded, showing an "Override by" field with the value "0.06" and a unit "L". Below this is a table with columns for width (W) and length (L) thresholds. The "Metal2" section is also expanded, showing an "Override by" field with the value "0.07" and a unit "L". Below this is another table with columns for width (W) and length (L) thresholds. The "Metal4", "Metal5", and "Metal6" sections are collapsed. At the bottom right, there are "OK" and "Cancel" buttons.

		0.005	0.325	0.755	1.505	2.505	3.505
W	0.005	0.06	0.06	0.06	0.06	0.06	0.06
	0.105	0.06	0.1	0.1	0.1	0.1	0.1
	0.755	0.06	0.1	0.25	0.25	0.25	0.25
	1.505	0.06	0.1	0.25	0.45	0.45	0.45
	2.505	0.06	0.1	0.25	0.45	0.75	0.75
	3.505	0.06	0.1	0.25	0.45	0.75	1.25

		0.005	0.325	0.755	1.505	2.505	3.505
W	0.005	0.07	0.07	0.07	0.07	0.07	0.07
	0.105	0.07	0.15	0.15	0.15	0.15	0.15
	0.755	0.07	0.15	0.25	0.25	0.25	0.25
	1.505	0.07	0.15	0.25	0.45	0.45	0.45
	2.505	0.07	0.15	0.25	0.45	0.75	0.75
	3.505	0.07	0.15	0.25	0.45	0.75	1.25

Important

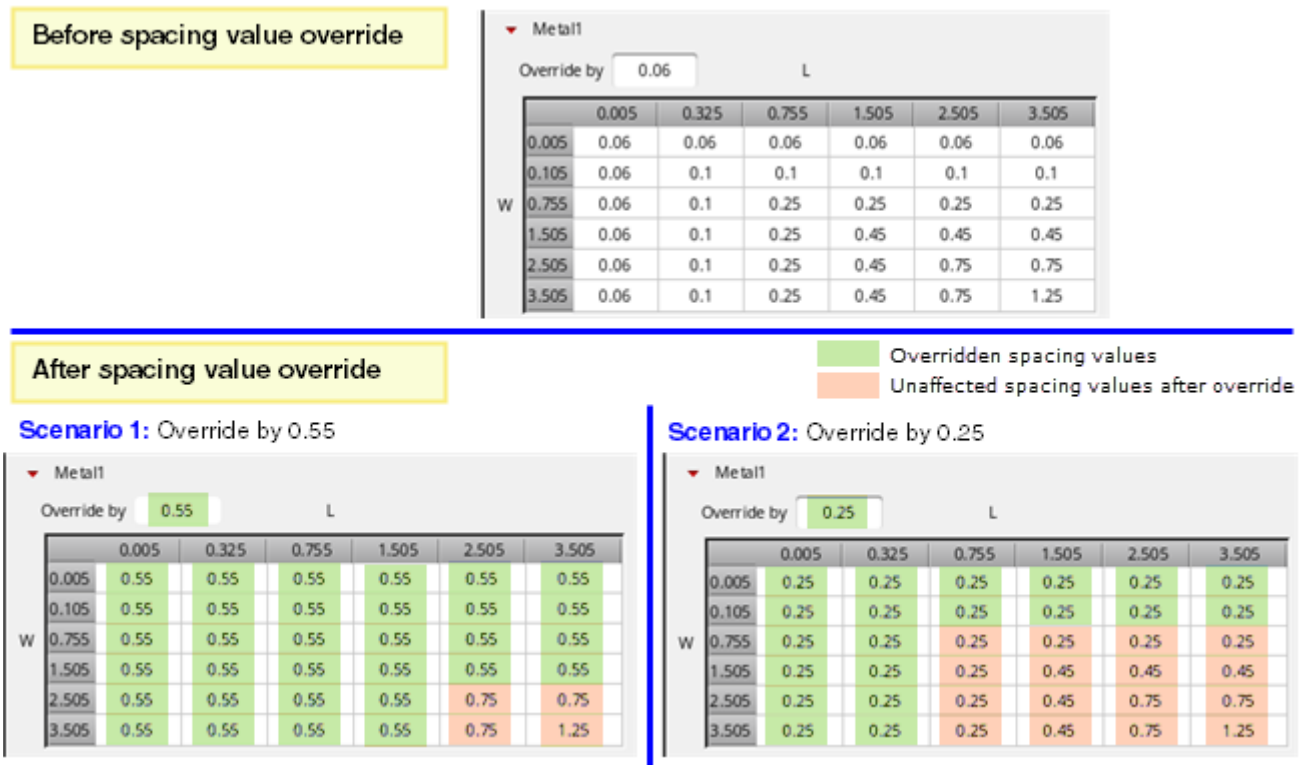
In the spacing table, you can edit only the spacing values. You cannot modify the width or length thresholds or the type of the spacing table. In two-dimensional spacing tables, the labels representing the rows and columns signify the type of the spacing table, that is, widthLength, widthWidth, or twoWidthParallelRunLength.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

The expanded section for each LPP has an *Override by* field where you can specify a single value by which the existing spacing values should be overridden. This field has the same impact on the table values as the [view/edit table](#) field.

Every time an override value is provided, it is compared with the reference table and based on that, only those table cell values are modified that are less than the override value. The reference table is obtained from the look-up order Design Process Rule Override --> technology Constraint Group --> Foundry. The figure below illustrates how this is done:



Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

field's border changes to red, as shown below, and the focus stays on it until you specify a valid value.

▼ Metal1

Override by L

	0.005	0.325	0.755	1.505	2.505	3.505
0.005	0.5	0.5	0.5	0.5	0.5	0.5
0.105	0.5	0.5	0.5	0.5	0.5	0.5
0.755	0.5	0.5	0.5	0.5	0.5	0.5
1.505	0.5	0.5	0.5	0.5	0.5	0.5
2.505	0.5	0.5	0.5	0.5	0.75	0.75
3.505	0.5	0.5	0.5	0.5	0.75	1.25

Pressing the `Enter` key after specifying a valid override value in the *Override by* field, replaces the table values that are less than the entered value. However, if the specified override value is invalid, the table values get reset to the default as per the technology database regardless of any previous valid override (as can be seen in the table of the figure above) and the *Override by* field gets reset, as shown below.

▼ Metal1

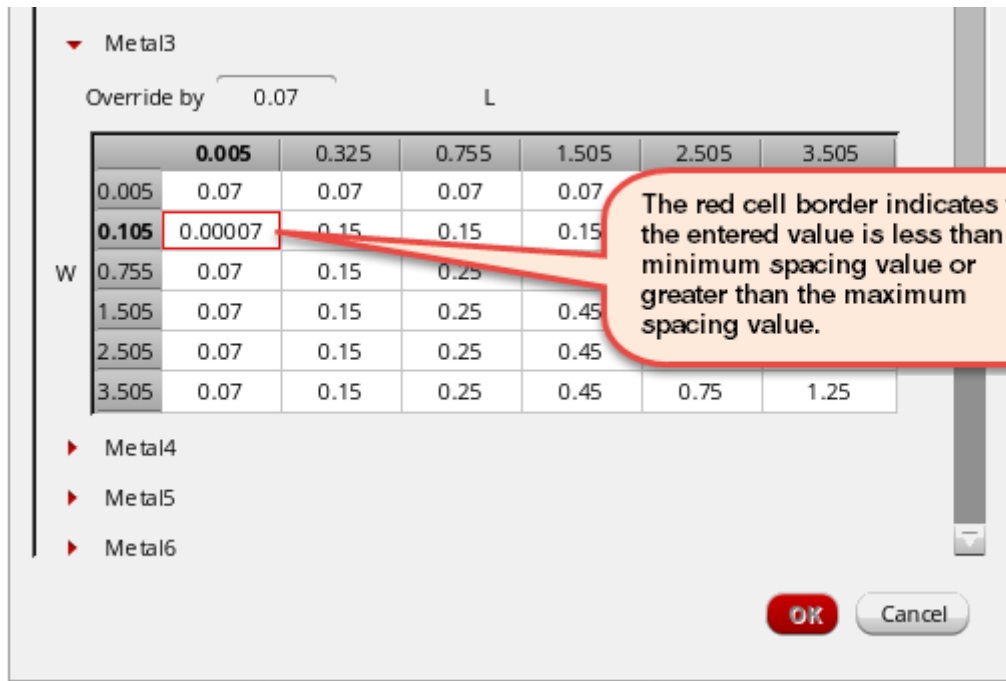
Override by 0.0 L

	0.005	0.325	0.755	1.505	2.505	3.505
0.005	0.06	0.06	0.06	0.06	0.06	0.06
0.105	0.06	0.1	0.1	0.1	0.1	0.1
0.755	0.06	0.1	0.25	0.25	0.25	0.25
1.505	0.06	0.1	0.25	0.45	0.45	0.45
2.505	0.06	0.1	0.25	0.45	0.75	0.75
3.505	0.06	0.1	0.25	0.45	0.75	1.25

In addition to overriding spacing values by specifying a single value as explained above, in the Layer Spacings form, you can edit an individual cell value in the spacing table. Each cell of the spacing table has its own minimum spacing value threshold. Therefore, while editing a cell in the spacing table, if you enter a value that is less than the minimum allowable spacing value or greater than the maximum allowable spacing value, the border of the edited cell changes to red and will remain like this until a valid value is entered.

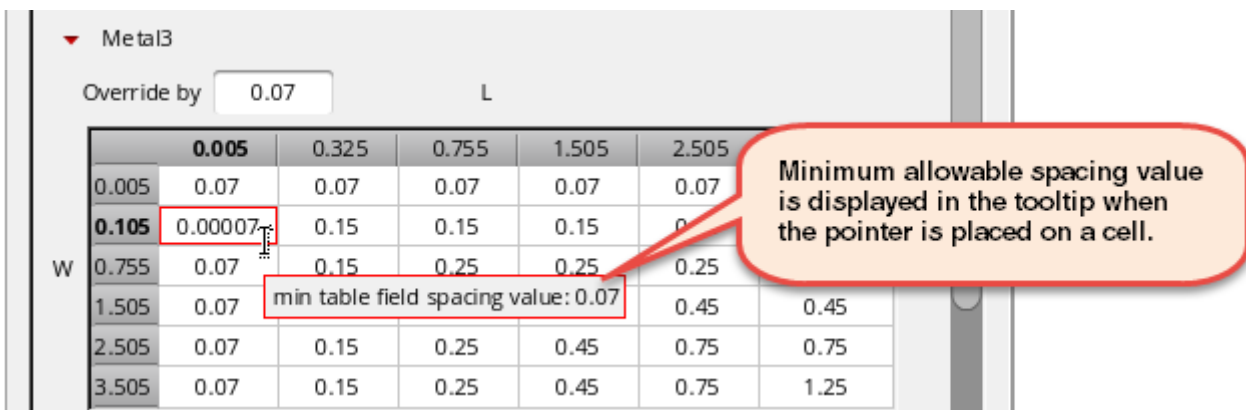
Virtuoso Unified Custom Constraints User Guide

Default Constraint Types



Note: The Layer Spacings form can be saved with invalid values.

When you place the pointer on a cell, a tooltip displays the minimum allowable spacing value, as shown below.



When you input a spacing value that is not an integral multiple of the manufacturing grid resolution specified in the technology database, the value is rounded off to the nearest

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

integral multiple. For example, if the specified manufacturing grid resolution is 0.005, then the expected spacing values are calculated as follows:

User Input	Expected Spacing Value
0.05	0.05
0.052	0.055
0.055	0.055
0.057	0.06

Note: The expected spacing values are rounded off to three decimal places.

The modified spacing value is not saved in the technology database. It is instead saved as a process rule override at the object level.

The spacing value can also be viewed or edited from the Process Rule Editor. Any updates are automatically synchronized between the Process Rule Editor and the layer spacing table in the Constraint Parameter Editor.

Keyboard Shortcuts Useful while Editing Values in the Spacing Table

Keyboard Keys	Use
Enter, up arrow, or down arrow	Shifts the focus from one cell to the other in the same column.
Tab	Shifts the focus to the next column in the same row.
Esc	Discards the edited spacing value. However, this works only as long as the focus is on the edited cell.

Suppose you modified the spacing table values or its semantics, that is, you added or removed rows or columns, or changed the table type by using the Process Rule Editor (PRE), such that the object-level constraint has no match with the reference table defined in the technology file or Design-Level Process Rule. You will see the following behavior:

- In the Constraint Parameter Editor,
 - The override value gets removed from the *view/edit table* cell and you are not allowed to specify a new override value.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

- ❑ The tooltip shows the spacing table saved from PRE when you move the pointer on the related LPP listed under the *Spacing* parameter.
- ❑ The tooltip shows a warning message about the inconsistency with respect to the reference table, as shown below:

The screenshot shows a table for the **Spacing** parameter. The table lists various materials and their spacing values. A tooltip is displayed over the **Metal1** row, showing a warning message about inconsistency with the reference table.

Spacing (various)	
Poly	0.06
Metal1	view/edit table
Metal2	view/edit table
Metal3	view/edit table
Metal4	view/edit table
Metal5	view/edit table
Metal6	view/edit table

Warning message to show an inconsistency between Process Rule Editor and a constraint's Spacing parameter

Inconsistency wrt Reference Table

		0.005	0.325	0.755	1.505	2.505	3.505
W	0.105	0.25	0.25	0.25	0.25	0.25	0.25
	0.755	0.25	0.25	0.25	0.25	0.25	0.25
	1.505	0.25	0.25	0.25	0.45	0.45	0.45
	2.505	0.25	0.25	0.25	0.45	0.75	0.75
	3.505	0.25	0.25	0.25	0.45	0.75	1.25

- In the Layer Spacing form,
 - ❑ The *Override by* field is not visible for the related LPP.
 - ❑ A warning message about the inconsistency with respect to the reference table is displayed, as shown below.

The screenshot shows the **Layer Spacings** form. The form lists various materials and their spacing values. A tooltip is displayed over the **Metal1** row, showing a warning message about inconsistency with the reference table.

The *Override by* field is not visible.

Inconsistent wrt Reference Table

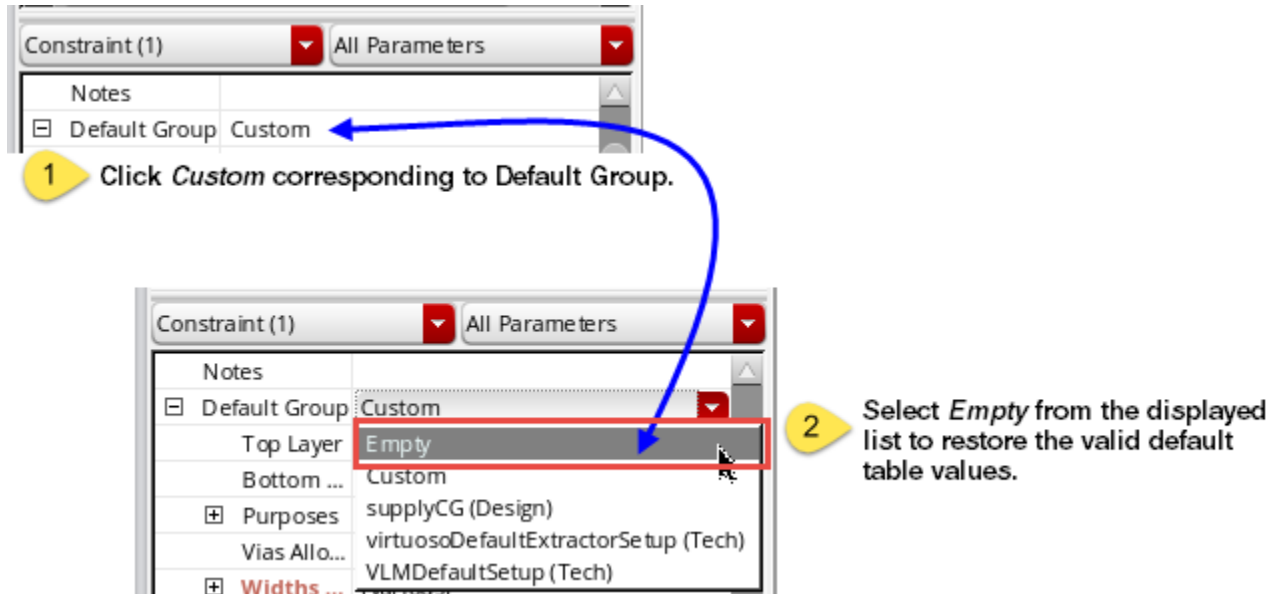
		0.005	0.325	0.755	1.505	2.505	3.505
W	0.105	0.25	0.25	0.25	0.25	0.25	0.25
	0.755	0.25	0.25	0.25	0.25	0.25	0.25
	1.505	0.25	0.25	0.25	0.45	0.45	0.45
	2.505	0.25	0.25	0.25	0.45	0.75	0.75
	3.505	0.25	0.25	0.25	0.45	0.75	1.25

The warning message shows that there is an inconsistency between the Process Rule Editor and the constraint's Spacing parameter.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

To discard the edits done by using PRE, change the value of the *Default Group* tab from *Custom* to *Empty*. This restores the valid default table values.



Parameter Types

The following constraint parameter types are available:

`int`

`bool`

`float`

`enum`

(one value from enumerated list)

`enumset`

(a set from an enumerated list)

`intRange`

`floatRange`

`string`

`string1DTbl`

Support of Shapes

It should be noted that the constraints system will only support shapes as constraint members if they have connectivity. That is, if they exist on a net. Currently, the constraint system will not support constraints on shapes, for any application, if they are not on a net.

Virtuoso Unified Custom Constraints User Guide

Default Constraint Types

Troubleshooting Constraints

This appendix provides information and guidance on the recommended use of the constraints management system. Its aim is to provide assistance and support in any areas where you may encounter unexpected results or difficulties.

If your concern is not addressed here, you should contact your local Cadence representative.

The following troubleshooting areas are covered in this appendix:

- [Performance Issues](#)
- [Read-Only Issues](#)
- [Transfer and Migration Issues](#)
- [Constraint Conflict Issues](#)

See also: [Virtuoso Unified Custom Constraints Known Problems and Solutions](#).

Performance Issues

Opening the Constraint Manager is Slow Even When No Constraints Exist

Note: This issue can effect the opening of workspaces that also contain the *Constraint Manager*.

This can occur because the constraints system looks for a `config.xml` file with constraint extensions in all directories in the Cadence search path (which includes the shell path).

A shorter path, or one that points to existing directories in the network, will reduce the lookup time considerably, therefore speeding up *Constraint Manager* opening time.

For more information on the Cadence search path, see Cadence Setup Search File: setup.loc in the *Cadence Application Infrastructure User Guide*.

Running a Circuit Prospector Finder is Slow When There Are Many Results

The highlighting of finder results using the Circuit Prospector can impact the speed. For cellviews with many results, you can improve the speed by setting the highlightCPResults environment variable to `nil`, which switches off the highlight feature.

Read-Only Issues

Closing Read-Only Schematic Does Not Ask for Constraints To Be Saved

When a read-only schematic window is closed, but not purged from memory, the *Constraint Manager* could also be closed even though it contains constraints that have not yet been saved. If this occurs you will be prompted to save any unsaved constraints upon exiting Virtuoso/CIW. Alternatively, you can save and close data ahead of exit by selecting *File – Close Data* from the CIW.

Constraints Shown as Read-Only When Schematic Is Opened in Write Mode

If a constraint view is read-only, or had previously been opened in read mode by another tool, the *Constraint Manager* reuses that data instead of forcing to re-open it in append mode. If this occurs, you can manually toggle the editing mode by selecting *Make Editable / Read Only* from the *Save* pull-down on Constraint Manager Toolbar.

Transfer and Migration Issues

Constraint Migration to Open Access (OA) Based Storage in IC612

When constraints are migrated to the new OA based storage system in IC612, the constraint view must be writable. If the view is not writable, errors will be produced by the OA design management layer.

How you approach this issue is largely dependent upon the importance of the constraints to be migrated:

- If the constraints are important you should make the constraint view writable to facilitate constraint migration.
- If the constraints are not of great importance, or you are content to just maintain them for potential future IC611 use, you should **MOVE** the entire view into, for example, a `constraint_611` directory.
- If you just want to save a copy of the constraints for potential future use in IC611, **COPY** the view into, for example, a `constraint_611` directory and also make the original constraint view writable to facilitate a successful migration to IC612 and beyond.

Constraints in Layout

To transfer/update constraints in the *Virtuoso Layout Suite XL*, select *Connectivity – Update – Layout Constraints*. This will transfer all the relevant constraints to the layout view.

- All single occurrence constraints will have a matching layout constraint, and all mFactor and iterated instance transfers will also be performed.
- All multiple occurrence (cell) constraints will have a matching layout constraint for *each* of the occurrences.

Note: You can cross-select between single occurrence constraints in the layout *Constraint Manager* with the occurrence of the cell constraint in the schematic *Constraint Manager*.

Missing All Constraints For a Schematic

If you switch between an ADE XL configuration and a physical configuration, the specified constraint view lists may differ. If this is the case, you should ensure that the content and order of the constraint views, in the new list to be used, is identical so that the intended constraint view will be used.

For example, for a schematic that has both “constraints_1” and “constraints_2” views, the first view in the list will be used. However, if your configuration has “constraints_2 constraints_1” as the constraint view list, the “constraints_2” view would be used.

Note: The same will be the case when schematics are used with the global constraint view list and a configuration is applied.

Constraint Conflict Issues

The constraint system identifies most common geometrical *over-constraints*; when a number of constraints share the same members. These *over-constraints* can cause conflicts in the design intent. Most constraint member and parameter inconsistencies are detected during constraint entry, when a consistency check is performed. It may however be beneficial to understand what potential inconsistencies can exist before entering constraints into a design, as it can help build a solid strategy when choosing an appropriate minimal constraint set.

Note: For more information on the constraint types mentioned in this section, see [Default Constraint Types](#).

Clusters

Clusters cannot have overlapping members. However, clusters can be hierarchical.

Modgen as Member of Other Constraints

In addition to existing as a constraint in its own right, a modgen can also be the member of the following constraint types:

- alignment
- distance
- symmetry
- matched orientation
- orientation
- guard ring

Modgen and Cluster

A modgen cannot be a member of a cluster constraint. However, all instance members of a modgen can be members of a cluster.

Modgen and Alignment

All modgen constraints contain embedded alignment constraints for each row and column.

Although it is possible to set an additional, and compatible, alignment constraint for the same members, it is preferable to choose only one constraint type (either modgen or alignment) for those same members.

Note: It is preferable to apply constraints to a modgen (for all constraints that support modgens) rather than to constrain the individual modgen members, as they could have implied constraints already by virtue of forming part of a modgen.

Modgen and Symmetry

All modgen constraints specify the order and orientation of each member inside the formed array. Although it is possible to set an additional, and compatible, symmetry constraint for a single, or number of the same members, it is preferable to choose only one constraint type (either modgen or symmetry) for those members.

Note: A symmetry constraint can be set against a modgen or a pair of modgens.

Two symmetrical instances can also be members of different modgens. However, if this is the case, then the two modgens must be symmetrical.

Note: It is preferable to apply constraints to a modgen (for all constraints that support modgens) rather than to constrain the individual modgen members, as they could have implied constraints already by virtue of forming part of a modgen.

Net Class

Net-based constraints that apply to groups of nets are not allowed to have common members that are shared with other constraints. For example, the following combinations of constraints are infeasible:

- Net Class and Diff Pair
- Net Class and Matched Length
- Net Class and Symmetry

If an overlap is detected, such constraints are not generated and a warning message is displayed in the CIW with the names of the common members.

Symmetry and Alignment

A symmetry constraint, defined for a pair of objects, will contain an embedded alignment constraint. The alignment axis will be orthogonal to the symmetry axis. It is therefore not required (and not recommended) to *over-constrain* a pair of symmetrical objects with an additional alignment constraint.

However, an alignment constraint can be valuable and its use is especially valid when aligning two, or more, pairs of symmetrical objects together.

Symmetry and Cluster

Note: This conflict is specifically related to a current limitation in the Analog Placer and Constraint Aware Editing.

If a symmetrical object is a member of a cluster, the other object must also be a member of the same cluster.

Symmetry and Orientation

Two symmetrical objects can both have an orientation constraint. However, the allowed orientation for these objects must be compatible with the value of the mirror parameter.

For example, if `mirror = true` the orientation of one symmetrical object must be the mirrored orientation of the other object of the same symmetrical pair.

Symmetry Axis

When several self-symmetrical objects, or pairs of symmetrical objects, have the same symmetry axis they will share the same symmetry axis for placement.

To have different groups of symmetry placement at least two different names of symmetry axis are required.

It is recommended that a specific axis be set for all symmetry and alignment constraints, as leaving the constraints with the default axis may lead to unpredictable results.

Note: The name, direction, and coordinate of each symmetry axis can be changed and fixed.

Environment Variables

This appendix covers the environment variables that can be used to control and customize the characteristics and behavior of custom constraints. Refer to the [List of Environment Variables](#) section for a complete list of environment variables available for use.

For information about the environment variables specific to:

- Virtuoso Schematic Editor, refer to the [Environment Variables](#) chapter in the *Virtuoso Schematic Editor User Guide*.
- Virtuoso Layout Suite XL, refer to the [Environment Variables](#) chapter in the Virtuoso Layout Suite documentation.

Setting Environment Variables

You can set the environment variables in the following three ways:

- Within the [.cdsenv File](#)
- Within the [.cdsinit File](#)
- In the [CIW](#)

.cdsenv File

Add environment variables to the `.cdsenv` file when the settings should be applied while launching the Constraint Manager.

Use the following syntax:

```
constraint environmentVariableName dataType value
```

For example:

```
constraint addParallelDevices boolean t
```

.cdsinit File

Like the `.cdsenv` file, the environment variable settings saved in the `.cdsinit` file get applied when you launch the Constraint Manager.

Use the `envSetVal()` command, which has the following syntax, to add environment variables to the `.cdsinit` file:

```
envSetVal("constraint" "environmentVariableName" 'dataType value)
```

For example:

```
envSetVal("constraint" "addParallelDevices" 'boolean t)
```



The *dataType* should be preceded with a single quote, else the command will not work. Also, if *dataType* is `string`, enclose its *value* within double quotes.

CIW

Use the `envSetVal()` command in the CIW to set an environment variable for the duration of the current session. The syntax is the same as described above for the `.cdsinit` file.

Note: Alternatively, to set environment variables for a single session, you can include the `envSetVal()` command in any Cadence SKILL file that you load.

Displaying the Current Value of an Environment Variable

To determine the current value of any Constraint Manager environment variable, use the following syntax in the CIW.

```
envGetVal("constraint" "environmentVariableName")
```

List of Environment Variables

The following environment variables are available for use:

- addParallelDevices
- autoConstraintNotes
- autoSelectTemplateConstraint
- comanageConstraintView
- comanageFollowModeOnDescend
- comanageFollowModeOnOpen
- consistencyFailureDefaultWarning
- constraintMenuHistorySize
- constraintNotesSyncOnDesignCheck
- constraintNotesSyncOnDesignOpen
- crossCoupleDiffPairRequireSourceNotSupply
- crossHalo
- currentMirrorIncludeDummies
- defaultViewName
- diffPairRequireSourceNotSupply
- disableCICustomCGPopup
- distanceMFactorExpansion
- editTransferredTemplates
- highlightCPResults
- keepFigGroupMembersWithinSameCluster
- matchedToIgnore
- maxMatchesPerDevice
- noteTooltipMaxCharWidth
- noteTooltipMaxLines

Virtuoso Unified Custom Constraints User Guide

Environment Variables

- purgeOnCloseCellView
- restrictNetGroupMembership
- updateConstraints
- useOldComparisonReport
- viewNameList
- viewWithConfigCell

addParallelDevices

Syntax

In the `.cdsenv` file:

```
constraint addParallelDevices boolean {t | nil}
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "addParallelDevices" 'boolean {t | nil})
```

Description

Controls whether parallel devices should be included in the structures during structure matching.

Default Value: `nil`

Example

Set the following to include the parallel devices while matching expressions for captured structures:

```
envSetVal("constraint" "addParallelDevices" 'boolean t)
```

autoConstraintNotes

Syntax

In the `.cdsenv` file:

```
constraint autoConstraintNotes boolean {t | nil}
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "autoConstraintNotes" 'boolean {t | nil})
```

Description

Enables the automatic creation of constraint notes.

When you set this environment variable to `t` and create new constraints, the constraint notes automatically get added to the schematic. In addition, when you change or delete any constraint, the constraint notes get updated.

Default Value: `nil`

Example

Set the following to enable the feature to automatically add constraint notes to the schematic when constraints are added, updated, and deleted:

```
envSetVal("constraint" "autoConstraintNotes" 'boolean t)
```

autoSelectTemplateConstraint

Syntax

In the `.cdsenv` file:

```
constraint autoSelectTemplateConstraint boolean {t | nil}
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "autoSelectTemplateConstraint" 'boolean {t | nil})
```

Description

Restores the original behavior that existed in IC6.1.5 where on selecting a template in the Constraint Manager, the Constraint Property Editor displayed all the parameters of all the constraints contained within the template.

In IC6.1.6, the template parameters are defined within the template definition itself. Therefore, the template parameters, if there are any, are displayed with the template. If the template does not have any parameters, then no parameters are displayed.

Note: Setting the environment variable to `t` displays only template parameters and not the constraint parameters.

Default Value: `nil`

Example

Set the following to display all parameters of all constraints associated with the selected template in the Constraint Property Editor:

```
envSetVal("constraint" "autoSelectTemplateConstraint" 'boolean t)
```

comanageConstraintView

Syntax

In the `.cdsenv` file:

```
schematic comanageConstraintView boolean {t | nil}
```

In the `.cdsinit` file or the CIW:

```
envSetVal("schematic" "comanageConstraintView" 'boolean {t | nil})
```

Description

Controls whether the schematic and constraints views are co-managed or managed separately when specific schematic editor commands are used. When set to `t`, schematic and constraint views are co-managed. When set to `nil`, they are handled separately. The default is `t`. This feature is available only in Schematics XL.

This feature applies to the following commands

- *Save, Save a Copy, Discard Edits, Check and Save, and Change Edit Mode*, in the *File* menu.
- *Check — Current Cellview*

Applying the commands to both schematic and constraint views simultaneously, saves you having to use separate commands to perform the same operation on the two views.

If set to `nil`, the listed commands apply to the schematic view only. Changes to the constraint view are not applied and instead the Constraint Manager assistant options: *Save Constraints*, *Save a Copy*, *Discard Edits*, and *Make Read Only* must be used instead.

If set to `t` and changes have only been made to the constraint view, the *File* menu options, *Save* and *Discard Edits* are displayed with the suffix (Constraint).

Default Value: `t`

Example

To manage the schematic and constraint views separately:

```
envSetVal("schematic" "comanageConstraintView" 'boolean nil)
```

comanageFollowModeOnDescend

Syntax

In the `.cdsenv` file:

```
schematic comanageFollowModeOnDescend boolean {t | nil}
```

In the `.cdsinit` file or the CIW:

```
envSetVal("schematic" "comanageFollowModeOnDescend" 'boolean {t | nil})
```

Description

Controls the edit mode of the constraint view on descend edit through the hierarchy.



This environment variable works in conjunction with the environment variable [comanageConstraintView](#).

Setting the `comanageFollowModeOnDescend` and `comanageConstraintView` environment variables to `t` ensures that a sub-cell's constraint view is opened in edit mode every time when you select the *Edit – Hierarchy – Descend Edit* menu option in Schematic XL to descend into the sub-cell from the top cell. However, with the *Edit – Hierarchy – Descend Read* menu option, the descended subcell's constraint view may not open in read mode if it was last opened for edit in the current session.

When this environment variable is set to `nil`, a sub-cell's constraint view might not necessarily open in edit mode despite using the *Edit – Hierarchy – Descend Edit* menu option. The constraint view can change to read only if it was last opened in read mode in the current session. Also, when you are [Working with Constraint Views in a Design-Managed Environment](#), the expected check-in and checkout behavior related to the edit mode operation is not followed for a constraint view.

Default Value: `t`

comanageFollowModeOnOpen

Syntax

In the `.cdsenv` file:

```
schematic comanageFollowModeOnOpen boolean {t | nil}
```

In the `.cdsinit` file or the CIW:

```
envSetVal("schematic" "comanageFollowModeOnOpen" 'boolean {t | nil})
```

Description

Controls whether the constraint view should be opened in editable or non-editable mode.



This environment variable works in conjunction with the environment variable [comanageConstraintView](#).

The default setting, `t`, of this environment variable mainly controls the edit mode of the constraint views when a top cell is opened in edit mode followed by its subcell. For example, if you open the Constraint Manager assistant for top cell, `inv_top`, in tandem with a subcell of this top cell, `inv_sub`. notice that the Constraint Manager assistant for `INV_sub1` also opens in edit mode.

If this environment variable is set to `nil` and you access the constraint view of a sub cell in tandem with the editable cellview of its top cell, then the subcell's constraint view may open in non-editable mode. Also, when you are [Working with Constraint Views in a Design-Managed Environment](#), the expected check-in and checkout behavior related to opening a constraint view is not followed.

Default Value: `t`

consistencyFailureDefaultWarning

Syntax

In the `.cdsenv` file:

```
constraint consistencyFailureDefaultWarning boolean {t | nil}
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "consistencyFailureDefaultWarning" boolean {t | nil})
```

Description

Controls the display of system-generated warning messages for consistency failures. By default, this environment variable disables the display of system-generated messages for custom constraints and system constraints that have corresponding custom verification callback. To enable the display of such messages, set this environment variable to `t`.

Default Value: `nil`

Example

To enable the display of system-generated warning messages for consistency failures, set the following:

```
envSetVal("constraint" "consistencyFailureDefaultWarning" boolean t)
```

constraintMenuHistorySize

Syntax

In the `.cdsenv` file:

```
constraint constraintMenuHistorySize int integer
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "constraintMenuHistorySize" 'int integer)
```

Description

Sets the size of the constraint history displayed at the end of the *Constraint Creation* drop-down list, which is under the last horizontal line, in the Constraint Manager. The history is a list of the last *N* constraints created and gives you quick access to them.

If set to 0, the feature of maintaining the constraint history gets disabled.

Default Value: 5

Example

To maintain the last 7 records in the constraint history, set the following:

```
envSetVal("constraint" "constraintMenuHistorySize" 'int 7)
```


constraintNotesSyncOnDesignCheck

Syntax

In the `.cdsenv` file:

```
constraint constraintNotesSyncOnDesignCheck boolean {t | nil}
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "constraintNotesSyncOnDesignCheck" 'boolean {t | nil})
```

Description

Enables the automatic synchronization of constraint notes when the design is checked.

By default, this environment variable is set to `nil`. When you set this environment variable to `t`, the constraint notes get synchronized on design check.

For more information, see [Setting Up Automatic Generation of Constraint Notes](#).

Default Value: `nil`

Example

To enable the feature to automatically synchronize the constraint notes when a design is checked, set the following:

```
envSetVal("constraint" "constraintNotesSyncOnDesignCheck" 'boolean t)
```

constraintNotesSyncOnDesignOpen

Syntax

In the `.cdsenv` file:

```
constraint constraintNotesSyncOnDesignOpen boolean {t | nil}
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "constraintNotesSyncOnDesignOpen" 'boolean {t | nil})
```

Description

Enables the automatic synchronization of constraint notes when the design is opened in edit mode.

By default, this environment variable is set to `nil`. Its setting to `t` causes the desired effect only when the `autoConstraintNotes` environment variable has also been set to `t` to enable automatic creations of constraint notes.

When you set this environment variable and `autoConstraintNotes` to `t`, the constraint notes get synchronized on opening a design in edit mode.

For more information, see [Setting Up Automatic Generation of Constraint Notes](#).

Default Value: `nil`

Example

To enable the feature to automatically synchronize the constraint notes on opening a design, set the following:

```
envSetVal("constraint" "constraintNotesSyncOnDesignOpen" 'boolean t)
```

crossCoupleDiffPairRequireSourceNotSupply

Syntax

In the `.cdsenv` file:

```
constraint crossCoupleDiffPairRequireSourceNotSupply boolean {t | nil}
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "crossCoupleDiffPairRequireSourceNotSupply"  
          'boolean {t | nil})
```

Description

When this environment variable is set to `t`, a cross-coupled differential pair structure will be recognized only if the source connection of the structure is not connected to a supply net. Setting the environment variable to `nil` removes this restriction.

Default Value: `t`

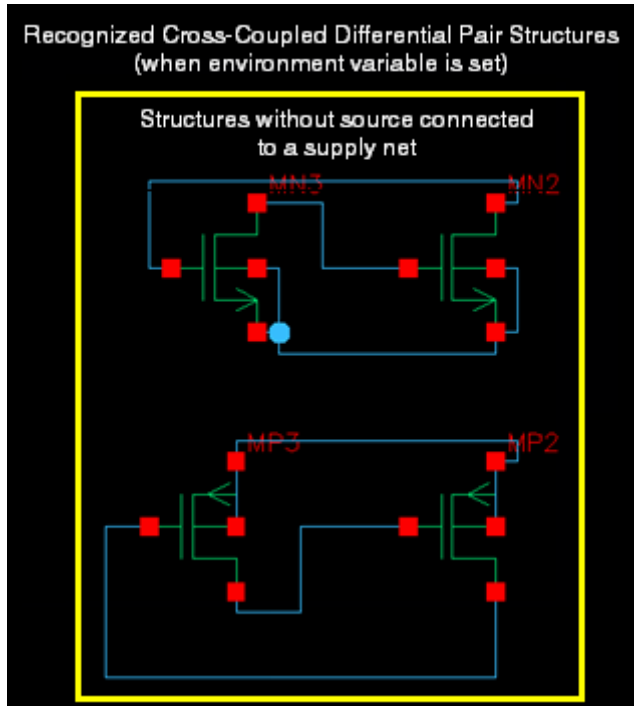
Example

When the `crossCoupleDiffPairRequireSourceNotSupply` environment variable is set, the following cross-coupled differential pair structures get recognized and listed on

Virtuoso Unified Custom Constraints User Guide

Environment Variables

selecting the MOS Differential Pair - Cross Coupled finder from the *Search for* drop-down list box in the Circuit Prospector assistant:



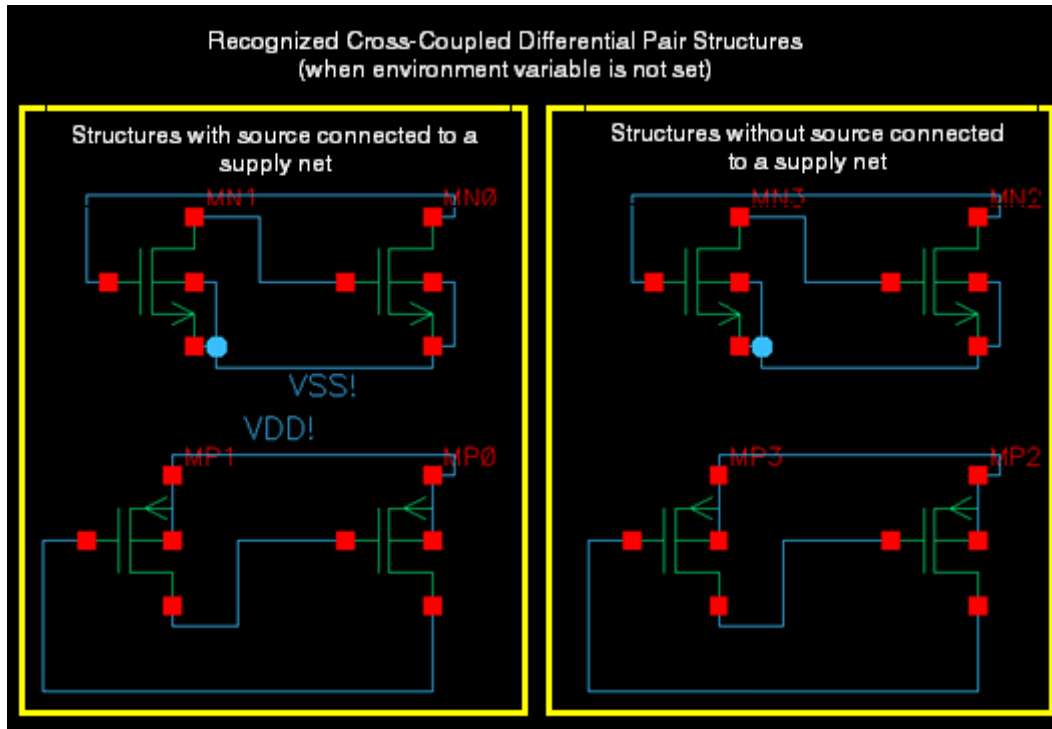
However, when you set the following to remove the restriction on the recognition of the cross-coupled differential pair structures:

```
envSetVal("constraint" "crossCoupleDiffPairRequireSourceNotSupply" 'boolean nil)
```

Virtuoso Unified Custom Constraints User Guide

Environment Variables

the following structures can be recognized (note the two additional structures on the right):



crossHalo

Syntax

In the `.cdsenv` file:

```
constraint crossHalo boolean {t | nil}
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "crossHalo" 'boolean {t | nil})
```

Description

Controls the cross-selection of constraints between the schematic *Constraint Manager* and the layout *Constraint Manager*, as defined in the constraint partition. When set to `t`, the cross-selection is enabled. Setting the environment variable to `nil` disables the cross-selection.

The following caveats must be considered while using the `crossHalo` environment variable:

- The *Constraint Manager* should be open in both schematic and layout.
Note: If the *Constraint Manager* is closed in either, the cross-selection will not work.
- Corresponding constraints should be defined in both schematic and layout *Constraint Manager* and linked to the schematic and layout views.

For more information, refer to the [Cross-Selection of Constraints Between Schematic and Layout Constraint Managers](#) section.

Default Value: `t`

Example

To disable the cross-selection of constraints, set the following:

```
envSetVal("constraint" "crossHalo" 'boolean nil)
```

currentMirrorIncludeDummies

Syntax

In the `.cdsenv` file:

```
constraint currentMirrorIncludeDummies boolean {t | nil}
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "currentMirrorIncludeDummies" 'boolean {t | nil})
```

Description

Controls whether dummy devices should also be recognized as part of the identified Current Mirror structure.

A dummy device has its source, drain, and bulk connected together and its gate connected to the gate of the reference device in the Current Mirror structure. By default, dummy devices are not included in the recognized Current Mirror structures. To allow dummy devices to be included, set this environment variable to `t`.

Default Value: `nil`

Example

To recognize the dummy devices as a part of the identified Current Mirror structure, set the following:

```
envSetVal("constraint" "currentMirrorIncludeDummies" 'boolean t)
```

defaultViewName

Syntax

In the `.cdsenv` file:

```
constraint defaultViewName string "<userDefinedName>"
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "defaultViewName" 'string "<userDefinedName>")
```

Description

Defines the name of the constraint view to use when opening Constraint Manager in a schematic. This default view name is also accessible by selecting *Options* from *Save (Constraints)*, which is the first menu in the Constraint Manager toolbar when accessed from the schematic.

Default Value: `constraint`

Example

To specify `constraint2` as the default view name, set the following:

```
envSetVal("constraint" "defaultViewName" 'string "constraint2")
```


diffPairRequireSourceNotSupply

Syntax

In the `.cdsenv` file:

```
constraint diffPairRequireSourceNotSupply boolean {t | nil}
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "diffPairRequireSourceNotSupply" 'boolean {t | nil})
```

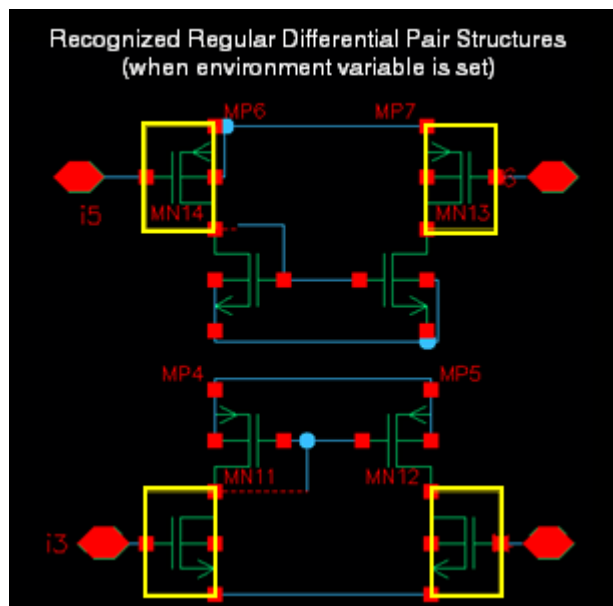
Description

When this environment variable is set to `t`, a Differential Pair structure will be recognized only if the source connection of the structure is not connected to a supply net. Setting the environment variable to `nil` removes this restriction.

Default Value: `t`

Example

When the `diffPairRequireSourceNotSupply` environment variable is set, the following differential pair structures get recognized and listed on selecting the MOS Differential Pair finder from the *Search for* drop-down list box in the Circuit Prospector assistant:



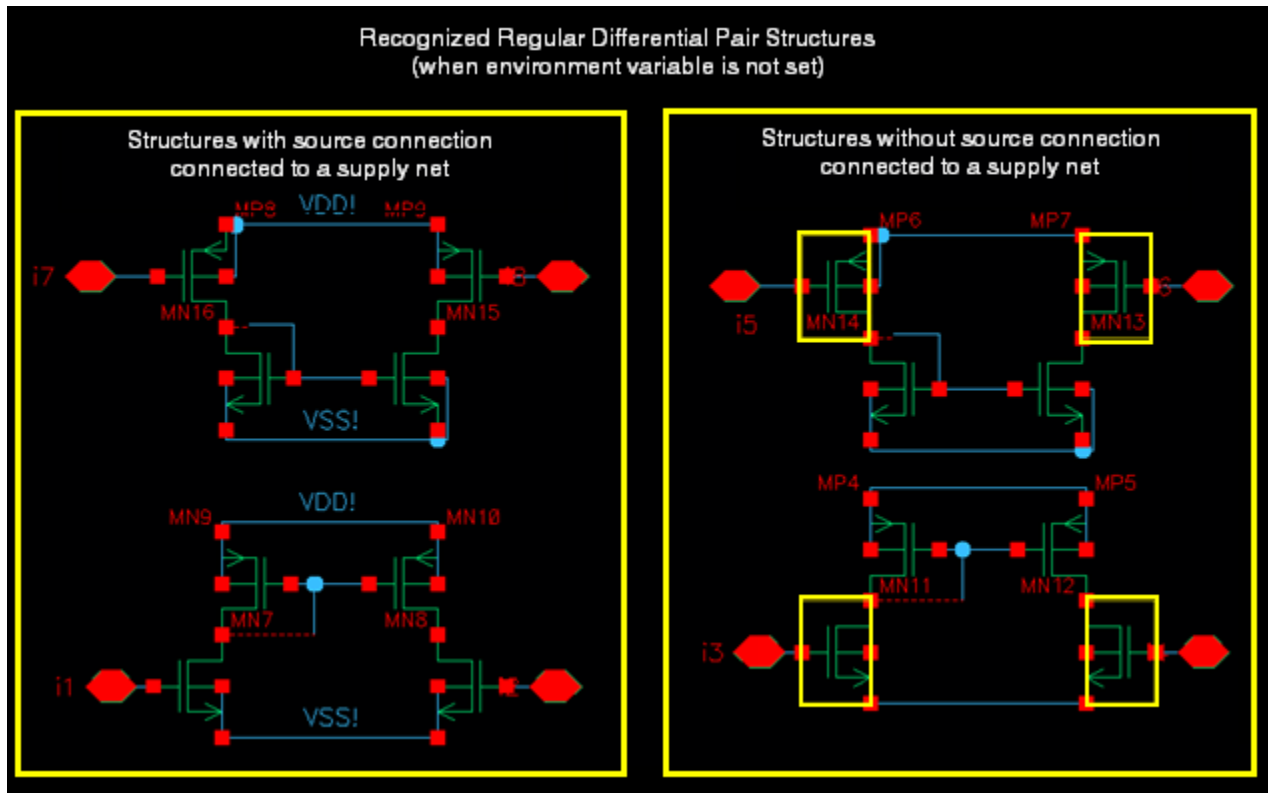
Virtuoso Unified Custom Constraints User Guide

Environment Variables

However, when you set the following to remove the restriction on the recognition of the differential pair structures:

```
envSetVal("constraint" "diffPairRequireSourceNotSupply" 'boolean nil)
```

the following structures can be recognized (note the two additional structures on the left):



disableCICustomCGPopup

Syntax

In the `.cdsenv` file:

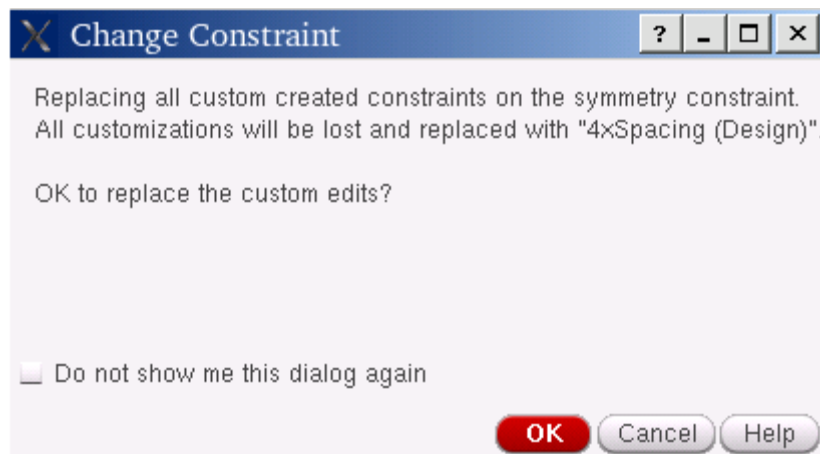
```
constraint disableCICustomCGPopup boolean {t | nil}
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "disableCICustomCGPopup" 'boolean {t | nil})
```

Description

When value of any of *Within Group*, *Default Group*, or *Group to Outside Group* properties is changed from *Custom* to some default constraint groups value, the following warning message box appears that requires user input:



Note: The properties listed above are available in the *Constraint Parameter Editor* pane.

If the `disableCICustomCGPopup` environment variable is set to `t`, this message box does not appear and changes take place without the warning. So, setting this environment variable to `t` is equivalent to selecting the *Do not show me this dialog again* check box in the message box.

Default Value: `nil`

distanceMFactorExpansion

Syntax

In the `.cdsenv` file:

```
constraint distanceMFactorExpansion string {"pair" | "firstToAll" |  
      "symmetricPair"}
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "distanceMFactorExpansion"  
      'string {"pair" | "firstToAll" | "symmetricPair"}')
```

Description

For constraints that have mfactored instances as members, this environment variable helps to identify how the members of corresponding constraints in layout should be created on transfer or update.

The `distanceMFactorExpansion` environment variable requires one of the following valid values based on the logic described for each:

"pair"	For using this value, in a schematic constraint, you must have equal number of members and their corresponding mfactors. This means, if you have one member with three mfactors, the layout constraints will be created only if you have another member that also has three mfactors.
"firstToAll"	For using this value, in a schematic constraint, the number of members and their mfactors need not be equal. For example, if you have a member with two mfactors and two other members with three mfactors each, only two constraint members will be created.
"symmetricPair"	For using this value, you must have equal number of members, but their corresponding mfactors need not be equal. The pairing for creating constraints will be done till the point the mfactors of the member with lowest number of mfactors is reached. The unpaired mfactors will be omitted from the constraint creation process. See the Example section below to understand how it works.

Default Value: `firstToAll`

Virtuoso Unified Custom Constraints User Guide

Environment Variables

Example

- Suppose the schematic constraint, `Const_1`, has two members `M1` and `M2`, and the mfactor for both the members is 3. So, if the `distanceMFactorExpansion` environment variable is set to `pair`, while representing constraints in layout on transfer or update, three corresponding constraints will be created with following members:

First Constraint Members: `M1.1 M2.1`

Second Constraint Members: `M1.2 M2.2`

Third Constraint Members: `M1.3 M2.3`

- Suppose the schematic constraint, `Const_2`, has three members: `M1` with mfactor 2, `M2` with mfactor 3, and `M3` with mfactor 3. So, if the `distanceMFactorExpansion` environment variable is set to `firstToAll`, while representing constraints in layout on transfer or update, the two corresponding constraints will be created with following members:

First Constraint Members: `M1.1 M2.1 M2.2 M2.3 M3.1 M3.2 M3.3`

Second Constraint Members: `M1.2 M2.1 M2.2 M2.3 M3.1 M3.2 M3.3`

- Suppose the schematic constraint, `Const_3`, has two members: `M1` with mfactor 2 and `M2` with mfactor 3. So, if the `distanceMFactorExpansion` environment variable is set to `symmetricPair`, while representing constraints in layout on transfer or update, the two corresponding constraints will be created with following members:

First Constraint Members: `M1.1 M2.1`

Second Constraint Members: `M1.2 M2.2`

The last mfactor, `M2.3`, of member `M2` will be ignored because it does not have a corresponding mfactor available in member `M1`.

editTransferredTemplates

Syntax

In the `.cdsenv` file:

```
constraint editTransferredTemplates boolean {t | nil}
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "editTransferredTemplates" 'boolean {t | nil})
```

Description

Makes all transferred templates editable or non-editable in one go.

Default Value: `t`

Example

To make all transferred templates non-editable, set the following:

```
envSetVal("constraint" "editTransferredTemplates" 'boolean nil)
```

highlightCPResults

Syntax

In the .cdsenv file:

```
constraint highlightCPResults boolean {t | nil}
```

In the .cdsinit file or the CIW:

```
envSetVal("constraint" "highlightCPResults" boolean {t | nil})
```

Description

Controls whether finder results are highlighted in the Circuit Prospector. If set to `nil`, the results are not highlighted. For cellviews with many results, you can improve speed by setting this variable to `nil`.

Default Value: `t`

Example

Prevents finder results from being highlighted.

```
envSetVal("constraint" "highlightCPResults" boolean nil)
```

keepFigGroupMembersWithinSameCluster

Syntax

In the .cdsenv file:

```
constraint keepFigGroupMembersWithinSameCluster boolean {t | nil}
```

In the .cdsinit file or the CIW:

```
envSetVal("constraint" "keepFigGroupMembersWithinSameCluster" boolean {t | nil})
```

Description

Prevents figGroup members from becoming part of multiple constraints. If set to `nil`, figGroup members can be part of multiple constraints.

Default Value: `t`

Example

Lets figGroup members be part of multiple constraints.

```
envSetVal("constraint" "keepFigGroupMembersWithinSameCluster" boolean nil)
```


matchedToIgnore

Syntax

In the `.cdsenv` file:

```
constraint matchedToIgnore string "<list_of_param_names>"
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "matchedToIgnore" 'string "<list_of_param_names>")
```

Description

Specifies the names of parameters that should be ignored by the Matched Parameters constraint check and in Constraint-Aware Editing Mode. The value is a list of parameter names separated by spaces.

Different parameter names may be specified for `matchedToIgnore` in Schematic XL and Layout XL.

Default Value: " " (null string)

Example

```
envSetVal(constraint matchedToIgnore 'string "param1 param2")
envSetVal(schematic matchedToIgnore 'string "param1 param2")
```

Schematic XL looks for a non-empty schematic `matchedToIgnore` value and, if found, uses it. Otherwise, it uses the more general constraint `matchedToIgnore` value.

```
envSetVal(layoutXL matchedToIgnore 'string "param1 param2")
```

Layout XL, similar to Schematic XL, uses layoutXL `matchedToIgnore` variable if it is non-empty. Otherwise, it uses the more general constraint `matchedToIgnore` value.

maxMatchesPerDevice

Syntax

In the `.cdsenv` file:

```
constraint maxMatchesPerDevice int <integer>
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "maxMatchesPerDevice" 'int <integer>')
```

Description

Controls the maximum number of matches that a device can appear in when the Circuit Prospector structure finders are run. For example, Diff Pair and Current Mirror.

Default Value: 1

Example

To ensure maximum 4 matches per device appear when the Circuit Prospector structure finders are run, set the following:

```
envSetVal("constraint" "maxMatchesPerDevice" 'int 4')
```

noteTooltipMaxCharWidth

Syntax

In the `.cdsenv` file:

```
constraint noteTooltipMaxCharWidth int <integer>
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "noteTooltipMaxCharWidth" 'int <integer>)
```

Description

Controls the maximum number of characters that can appear in the line width of a tooltip displayed for the Constraint Notes.

Default Value: 30

Example

To ensure that maximum 50 characters can appear in a line width of a constraint note's tooltip, set the following:

```
envSetVal("constraint" "noteTooltipMaxCharWidth" 'int 50)
```

noteTooltipMaxLines

Syntax

In the `.cdsenv` file:

```
constraint noteTooltipMaxLines int <integer>
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "noteTooltipMaxLines" 'int <integer>)
```

Description

Controls the maximum number of lines that should be displayed in a tooltip for the available Constraint Notes.

Default Value: 10

Example

To ensure that maximum 15 lines appear in a constraint note's tooltip, set the following:

```
envSetVal("constraint" "noteTooltipMaxLines" 'int 15)
```

purgeOnCloseCellView

Syntax

In the `.cdsenv` file:

```
constraint purgeOnCloseCellView boolean {t | nil}
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "purgeOnCloseCellView" 'boolean {t | nil})
```

Description

If set to `t`, then on purging a schematic, the associated constraint view is also purged. If set to `nil`, the associated constraint view is not purged on purging the schematic.

Default Value: `t`

Example

To purge only the schematic and not the associated constraint view, set the following.

```
envSetVal("constraint" "purgeOnCloseCellView" 'boolean nil)
```

restrictNetGroupMembership

Syntax

In the `.cdsenv` file:

```
constraint restrictNetGroupMembership boolean {t | nil}
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "restrictNetGroupMembership" 'boolean {t | nil})
```

Description

If set to `t`, then while performing consistency checks for the `netGroup` type constraints, it is made sure that no member of the constraint being checked is part of any other `netGroup` including its own type.

Note: Currently, a `netGroup` constraint can be one of the following types: Net Bus, Diff Pair, Matched Length, Net Class, Strand, or Net Symmetry.

If set to `nil`, then:

- For Diff Pair, Matched Length, and Strand, this member consistency check only verifies that no members should be part of the same constraint type.
- For Net Class and Net Bus, this member consistency check verifies that no members should be part of Net Class or Net Bus type constraints.
- For Net Symmetry, there are no such member consistency checks.

Default Value: `t`

updateConstraints

Syntax

In the `.cdsenv` file:

```
constraint updateConstraints cyclic {automatic | selected | none}
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "updateConstraints" 'cyclic {automatic | selected | none})
```

Description

Controls the use model of the *Update Schematic/Layout Constraints* options. The following values can be set with `updateConstraints`:

Value	Use
<code>automatic</code>	If any constraints are selected in the source or target <i>Constraint Manager</i> , then those selected constraints will be updated when the <i>Update Schematic/Layout Constraints</i> option is selected. If no constraints are specifically selected then all constraints will be automatically updated.
<code>all</code>	All constraints will be updated regardless of what is currently selected when the <i>Update Schematic/Layout Constraints</i> option is selected.
<code>selected</code>	Only those currently selected constraints will be updated when the <i>Update Schematic/Layout Constraints</i> option is selected. No constraints will be updated if there are no currently selected constraints.

Default Value: `automatic`

Example

The `updateConstraints` variable can be set as follows:

■ In `.cdsenv`:

```
envGetVal("constraint" "updateConstraints" 'cyclic "automatic")
```

Virtuoso Unified Custom Constraints User Guide

Environment Variables

■ In `.cdsinit` or the CIW:

```
envGetVal("constraint" "updateConstraints" 'cyclic)  
envSetVal("constraint" "updateConstraints" 'cyclic "selected")
```


Virtuoso Unified Custom Constraints User Guide

Environment Variables

useOldComparisonReport

Syntax

In the `.cdsenv` file:

```
constraint useOldComparisonReport boolean {t | nil}
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "useOldComparisonReport" 'boolean {t | nil})
```

Description

Specifies to load the old format of the Constraint Comparison Report instead of the new default format introduced in IC6.1.6 ISR9 and ICADV12.1 ISR11.

By default, the `useOldComparisonReport` environment variable is set to `nil`. As a result, when you click the *Constraint Comparison Report* () button from the Constraint Manager assistant toolbar in Layout XL, the system launches a web browser and loads a report of the following format that compares the constraints existing in the current Schematic XL and Layout XL views.

					Last save date	Status	Revision	
Constraint Comparison Report					Fri Sep 19 09:15:09 2014			
ether_adc45n/adc_sample_hold:schematic					Mon May 16 15:45:01 2011	Not Modified	Not Managed	
ether_adc45n/adc_sample_hold:layout_routed1Test					Mon Sep 1 16:27:55 2014	Not Modified	Not Managed	
ether_adc45n.adc_sample_hold:constraint					Tue Jun 12 14:41:46 2012	Modified	Not Managed	
Status	Constraint Type	+	=	#	Name	Type	ether_adc45n adc_sample_hold schematic	ether_adc45n adc_sample_hold layout_routed1Tes
Violated	Distance	+	=				Distance.00	_ Distance.0
Violated and Not Bound	Alignment	+		#				ixVcpBoundaryTop
Violated and Not Bound	Matched Length	+		#				Constr_9
Violated and Not Bound	Orientation	+		#				_Constr_32
Violated and Not Bound	Symmetry	+		#				Constr_2
Mismatch	Guard Ring	+	=	#			_GuardRing_Modgen_3	_GuardRing_Modgen_
Mismatch	Guard Ring	+	=	#			_GuardRing_Modgen_4	_GuardRing_Modgen_
Mismatch	Modgen	+	=	#			_Constr_8	_Constr_8
Mismatch	Modgen	+	=	#			_Modgen_1	_Constr_21
Mismatch	Process Rule Overrides	+	=	#			CG_25	CG_72
Mismatch	Symmetry	+	=	#			_Constr_23	_Constr_23
Mismatch	Symmetry	+	=	#			_Constr_24	_Constr_24
Mismatch	Symmetry	+	=	#			_Constr_28	_Constr_28
Mismatch	Symmetry	+	=	#			Constr_29	Constr_29

Virtuoso Unified Custom Constraints User Guide

Environment Variables

However, if you prefer the old report format shown below or your existing test cases use it, you can switch back to the old comparison report by setting the `useOldComparisonReport` environment variable to `t`.



Default Value: nil

viewNameList

Syntax

In the `.cdsenv` file:

```
constraint viewNameList string "<list_of_view_names>"
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "viewNameList" 'string "<list_of_view_names>")
```

Description

Specifies a list of constraint view names. Use this environment variable in one of the following conditions:

- The schematic was not opened with a config or physConfig file.
- The config or physConfig file does not have a constraint view list.

Default Value: " " (null string)

Example

The following sets the `constraint` and `constraint2` as the view names:

```
envSetVal("constraint" "viewNameList" 'string "constraint constraint2")
```

Alternatively, you can use any one of the following:

```
constraint viewNameList 'string "constraint constraint2"
```

OR

```
constraint defaultViewName 'string "constraint"
```

viewWithConfigCell

Syntax

In the `.cdsenv` file:

```
constraint viewWithConfigCell boolean {t | nil}
```

In the `.cdsinit` file or the CIW:

```
envSetVal("constraint" "viewWithConfigCell" 'boolean {t | nil})
```

Description

Controls which constraint view out of the following should be used:

- the constraint view located in the same library directory as the schematic view.
- OR
- the constraint view located in the same library directory as the physConfig view.

If the schematic and layout views are in the same library directory, setting of this variable does not have any impact. The Boolean value set for the `viewWithConfigCell` environment variable is of importance only when the schematic and layout views are located in different libraries and are bound to a physConfig view.

Default Value: `nil`

Example

To ensure that the constraint view is stored with the config cell, set the following:

```
envSetVal("constraint" "viewWithConfigCell" 'boolean t)
```

Virtuoso Unified Custom Constraints User Guide

Environment Variables

The following figure shows which constraint view is used based on the status of the `viewWithConfigCell` environment variable:

