
Virtuoso Photonics Solution Guide

Product Version IC23.1

August 2023

© 2023 Cadence Design Systems, Inc.

Printed in the United States of America.

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

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

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

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

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

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

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

Contents

<u>Introduction to the EPDA Framework</u>	5
<u>EPDA for Photonics Integrated Circuits</u>	5
<u>About the CurvyCore Technology</u>	7
<u>Setting Environment Variables</u>	7
<u>Setting Environment Variables in a .cdsenv or .cdsinit File</u>	7
<u>Setting Environment Variables in the CIW</u>	8
 <u>2</u>	
<u>Using EPDA in the Virtuoso Environment</u>	9
<u>Specifying Technology Information</u>	9
<u>Launching Virtuoso in Photonics Mode</u>	10
<u>Launching the Virtuoso Photonics Option</u>	10
<u>Launching the Virtuoso Photonics Platform</u>	10
<u>Verifying the Photonics License</u>	10
<u>Generating Layout</u>	11
<u>Opening the Library Manager</u>	12
<u>Creating a New Library</u>	13
<u>Creating a New Library Cellview</u>	17
<u>Creating a Schematic Cellview</u>	18
<u>Opening a Schematic Cellview</u>	21
<u>Creating a Layout Cellview</u>	21
<u>Specifying the Components to be Generated</u>	22
<u>Specifying the I/O Pins to be Generated</u>	23
<u>Generating Selected From Layout</u>	25
<u>Abutting Photonic Elements</u>	28
<u>Generating Optical Chains</u>	29
<u>Generating an Incremental Chain</u>	31
<u>Generating an Anchored Chain</u>	31
<u>Editing Layout</u>	32
<u>Editing the Layout Parameters</u>	32
<u>Editing the Composite Waveguides</u>	33
<u>Managing the Layout Constraints</u>	37
<u>Routing Layout</u>	37

Virtuoso Photonics Solution Guide

<u>Verifying Design</u>	38
<u>Checking a Layout Against a Schematic</u>	38
<u>Checking XL Compliance</u>	39

3

<u>CurvyCore Building Blocks</u>	41
--	----

<u>CurvyCore Mathematical Objects</u>	43
---	----

<u>Curves</u>	44
---------------------	----

<u>Paths and Facets</u>	50
-------------------------------	----

<u>Attributes of ccPath Objects</u>	51
---	----

<u>Surfaces and Facets</u>	52
----------------------------------	----

<u>CurvyCore Pcells</u>	62
-------------------------------	----

<u>Waveguide Connectors</u>	67
-----------------------------------	----

<u>Straight Connectors</u>	67
----------------------------------	----

<u>Bend Connectors</u>	68
------------------------------	----

<u>Curve Connectors</u>	69
-------------------------------	----

<u>Waypoint Connectors</u>	70
----------------------------------	----

<u>Photonics Environment Variables</u>	73
--	----

<u>Setting Environment Variables</u>	73
--	----

<u>List of Photonics Solution Environment Variables</u>	74
---	----

<u>cweAlignElements</u>	76
-------------------------------	----

<u>cweMatchWidths</u>	78
-----------------------------	----

<u>opticalSigTypePropagation</u>	80
--	----

<u>opticalNetColor</u>	81
------------------------------	----

<u>opticalNetColoring</u>	82
---------------------------------	----

<u>opticalNetLineStyle</u>	83
----------------------------------	----

<u>phoAbutAutoAdjust</u>	84
--------------------------------	----

<u>phoAbutClass</u>	85
---------------------------	----

<u>phoAbutFunction</u>	86
------------------------------	----

<u>phoAbutNonPcells</u>	87
-------------------------------	----

<u>phoComposeMaster</u>	88
-------------------------------	----

<u>phoComposeMasterPromptOff</u>	89
--	----

<u>phoPinInputAngle</u>	90
-------------------------------	----

<u>phoPinLayer</u>	91
--------------------------	----

<u>phoPinRadius</u>	92
---------------------------	----

Virtuoso Photonics Solution Guide

<u>phoPinWidth</u>	93
<u>photonicDisplay</u>	94
<u>photonicPinWidth</u>	95
<u>photonicPinAngle</u>	96
<u>photonicPinFacetInPacket</u>	97
<u>photonicPinFacetOutPacket</u>	98
<u>photonicPinLabelPacket</u>	99
<u>photonicPinRadius</u>	100
<u>photonicPinRadiusLinePacket</u>	101
<u>photonicPinWidthLinePacket</u>	102
<u>srcOpticalElectricalConnection</u>	103
<u>srcOpticalMultiToSingle</u>	104
<u>srcOpticalSingleToMulti</u>	105
<u>srcOpticalTooManyConnections</u>	106

B

<u>Forms</u>	107
<u>Virtuoso Photonics Solution Forms</u>	107
<u>Composite Waveguide Editor Form</u>	108
<u>Generate Selected From Layout Form</u>	110

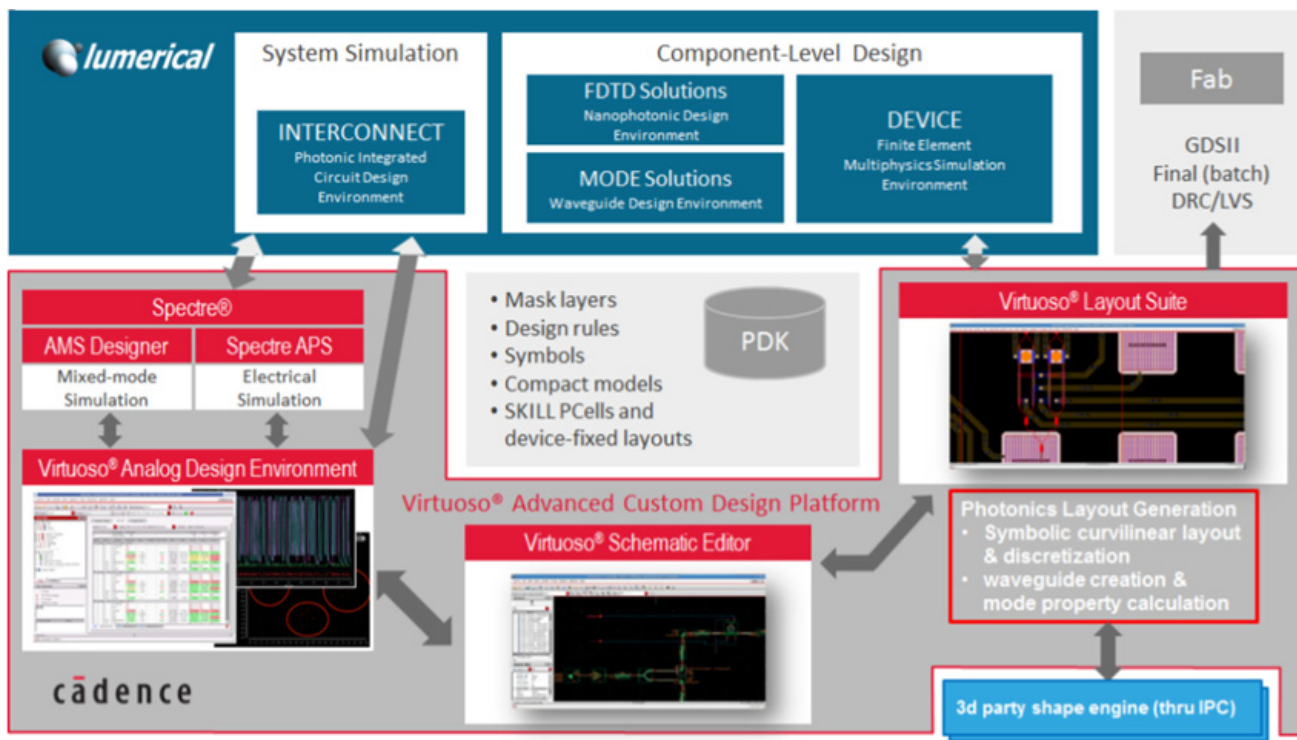
Virtuoso Photonics Solution Guide

Introduction to the EPDA Framework

Photonics, the science and technology of generating, controlling, and detecting light, is quickly moving into mainstream electronic designs. This is particularly true for communications hardware, where bandwidth demands are so high that only Photonic Integrated Circuits (PICs) offer a viable solution. Other key application areas include data centers, antenna and RF systems, biophotonics, and environmental sensing systems.

To address the challenges of designing PICs, Cadence® has developed an integrated Electronic-Photonic Design Automation (EPDA) environment in collaboration with Lumerical®.

EPDA for Photonics Integrated Circuits



Virtuoso Photonics Solution Guide

Introduction to the EPDA Framework

Built on the Cadence® Virtuoso® custom IC Design platform, the EPDA environment supports both the monolithic and hybrid design approaches. In monolithic approach, a single chip carries both traditional electronic and photonic design elements. In hybrid approach, a 3D-IC stack is used with a traditional electronics chip on top of a photonics chip, providing schematic and layout-driven design flows that support:

- Photonic schematic capture in the Cadence® Virtuoso® Schematic Editor.
- Photonic circuit simulation in the Cadence® Virtuoso® Analog Design Environment, using Lumerical® INTERCONNECT®, a dedicated PIC simulation engine.
- Photonic layout implementation in the Cadence® Virtuoso® Layout Suite environment.
 - ❑ Schematic-driven layout, using the same golden schematic as the one used for simulation.
 - ❑ Complex photonic Pcells and advanced photonic layout generators using the Cadence® SKILL® scripting language.
 - ❑ Backannotation of device and compounded waveguide parameters to schematic for layout-accurate optical re-simulation.
- Photonic component parameter and model generation for custom-defined components.
- Co-design of the electronic and photonic components for hybrid systems.

Prerequisites for Designing Photonics Integrated Circuits

As a developer and designer of Photonic Integrated Circuits (PIC) using Cadence technology, you should be familiar with:

- The Virtuoso design environment and application infrastructure mechanisms supporting consistent operations between all Cadence tools.
- The applications for designing and developing integrated circuits in the Virtuoso design environment, notably the Cadence® Virtuoso® Schematic Editor, Cadence® Virtuoso® Analog Design Environment, and the Cadence® Virtuoso® Layout Suite XL (Layout XL) layout editor.
- Virtuoso technology data.
- Component description format (CDF), which lets you create and describe components for use with Layout XL.

About the CurvyCore Technology

To support the development of complex curvilinear shapes for photonic designs, which are based on complicated mathematical equations, Cadence natively supports the CurvyCore[®] technology in Virtuoso[®] Layout Suite EXL. Integration of the CurvyCore technology in the Virtuoso custom IC design platform makes it possible for designers to work on complex technologies and designs in a familiar design environment. In addition, the collaboration makes it possible for designers to factor in both the electronic- and opto-electronic effects into the design much earlier in the cycle and put together a full electronic-photonic solution within the same design environment. For more information on the CurvyCore technology and how integration in Virtuoso makes it advantageous for a photonics designer, see [CurvyCore](#).

Setting Environment Variables

Environment variables control the Photonics design environment. For a list of all the environment variables supported by the Virtuoso Photonics Solution and the default and supported values of these environment variables, see [List of Photonics Solution Environment Variables](#).

There are two ways in which you can set environment variables:

- To set an environment variable that is applied every time you invoke the Virtuoso Photonics design environment, add the setting to your `.cdsenv` or `.cdsinit` file. For more information, see [Setting Environment Variables in a .cdsenv or .cdsinit File](#).
- To set an environment variable that is applied for the duration of the current session, use the `envSetVal()` command in the CIW. For more information, see [Setting Environment Variables in the CIW](#).

Setting Environment Variables in a .cdsenv or .cdsinit File

To have your environment variables set automatically when you start the Virtuoso Photonics design environment, do one of the following:

- Include the environment variables in the `.cdsenv` file in your home directory; for example,

```
layoutXL phoAbutNonPcells boolean t
```

- Include an `envSetVal()` command in your `.cdsinit` file

```
envSetVal("layoutXL" "phoAbutNonPcells" 'boolean t)
```

For more information on the `.cdsenv` and `.cdsinit` files, see [Environment Variables in the Virtuoso Layout Suite XL: Basic Editing User Guide](#).

Setting Environment Variables in the CIW

If you use any environment variable values consistently and do not want to set these values each time you use a command, you can set the variables to the value you normally use in the CIW and it will remain valid for the duration of the current session.

To set environment variables for a single session, do one of the following.

- Include `envSetVal ()` in any other Cadence® SKILL® file you load.
- Type `envSetVal ()` in the CIW.

For example, to set the `phoAbutNonPcells` variable, type the following in the CIW or include it in a setup file.

```
envSetVal("layoutXL" "phoAbutNonPcells" 'boolean t)
```

To determine the current value of any environment variable, type the following in the CIW.

```
envGetVal("layoutXL" "phoAbutNonPcells)
```

Note: Use the appropriate tool partition for the environment variable you want to set or look up the value for. For information about the tool partition—`layoutXL`, `graphic`, or `schematic`—associated with a specific environment variable supported for the Virtuoso Photonics Solution, see the related environment variable [documentation](#).

Related Information

- [Virtuoso Layout Suite XL: Basic Editing User Guide](#)
- [Virtuoso Layout Suite XL: Constraint Aware Editing User Guide](#)
- [Virtuoso Design Environment SKILL Reference](#)
- [Virtuoso Layout Suite SKILL Reference](#)

Using EPDA in the Virtuoso Environment

The unified Electronics-Photonics Design Environment (EPDA) framework is built upon the Virtuoso® Design Framework, leveraging the three core applications of the design framework—Virtuoso® Schematic Editor, Virtuoso® Analog Design Environment, and Virtuoso® Layout Suite. To enable a true co-simulation methodology within the Virtuoso Studio, the EPDA framework supports optical signal simulation in Cadence® Virtuoso® Analog Design Environment using the Lumerical® Interconnect Photonic Integrated Circuit (PIC) simulator. For more information about the Lumerical PIC simulator, visit the [Lumerical](#) website.

The key advantage of using the EPDA framework for running the Virtuoso® Photonics® Solution is that Virtuoso supports the integration and implementation of optical signals largely in the same way as that for electrical signals. This makes it easy for both existing and new users to use Virtuoso for creating their electro-optical designs.

Specifying Technology Information

To enable the generation and update of optical pins, you can update the technology file to define an additional constraint group—`virtuosoDefaultOpticalPinSetup`.

If specified, the `virtuosoDefaultOpticalPinSetup` constraint group defines the optical layers to use for generating and updating the optical pins. You can use the `opticalPinSetup` environment variable to specify the name of the constraint group to use for optical pin generation.

Note: You can also define the `virtuosoDefaultElectricalPinSetup` constraint group to specify the electrical layers to use for generating and updating electrical pins. To control the constraint group to use for electrical pin generation, use the `electricalPinSetup` environment variable.

If the `virtuosoDefaultOpticalPinSetup` or the `virtuosoDefaultElectricalPinSetup` constraint group is not defined, Virtuoso uses the `validLayers` defined in the default Layout EXL constraint group—`virtuosoDefaultSetup`—for pin generation.

Launching Virtuoso in Photonics Mode

Depending on whether or not you already have the Virtuoso licenses available, you can launch Photonics using one of the following licensing options:

- Launching the Virtuoso Photonics Option
- Launching the Virtuoso Photonics Platform

Note: Virtuoso Photonics Solution, irrespective of the license you choose, is supported only in Layout EXL.

Launching the Virtuoso Photonics Option

If you already have access to the Virtuoso design environment, you can invoke the additional Photonics capabilities supported in Virtuoso by using the Virtuoso Photonics Option (VPO) license.

To check out the Virtuoso Photonics Option license, set the following shell environment variables:

```
setenv Virtuoso_Photonics_Option t
setenv Virtuoso_MultiTech t
```

Launching the Virtuoso Photonics Platform

If you do not already have a dedicated license to access the Virtuoso design environment, you can check out a single, platform-level license called the Virtuoso Photonics Platform (VPP). This license enables all the Virtuoso design environment capabilities required for running the Photonics solution in addition to enabling the Virtuoso Photonics Solution.

To check out the Virtuoso Photonics Platform license, set the following shell environment variable:

```
setenv Virtuoso_Photonics_Platform t
```

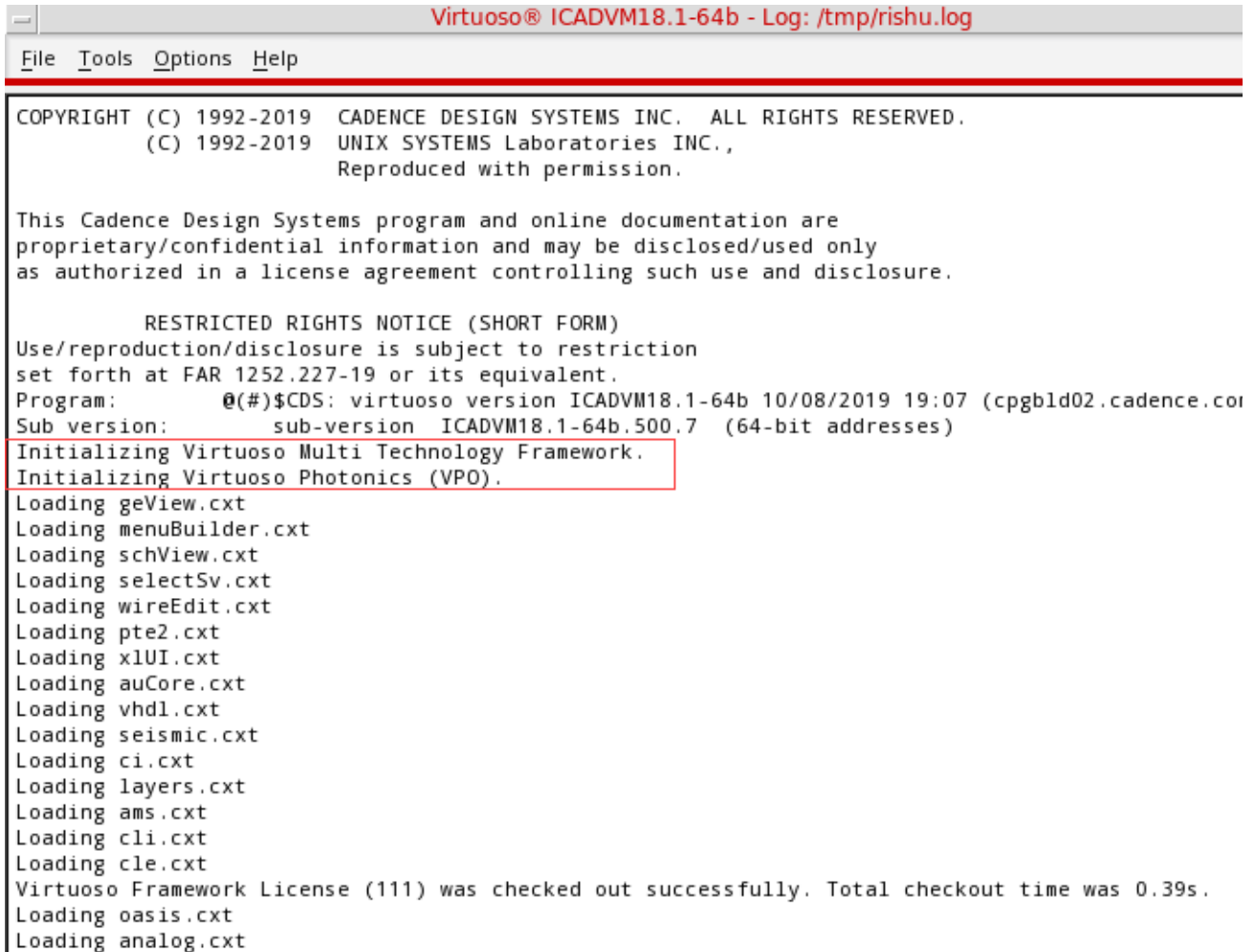
Verifying the Photonics License

To verify whether the required Virtuoso Photonics Solution licenses are checked out and that the EPDA framework is being enabled, check that the Command Interpreter Window (CIW) issues the confirmation messages as highlighted in the screenshot below. The CIW is the first window that opens when a Virtuoso session is launched.

Virtuoso Photonics Solution Guide

Using EPDA in the Virtuoso Environment

Note: Alternatively you can look for these confirmatory messages in the Virtuoso log file, which is saved by default as `CDS.log` in the home directory.



```
Virtuoso® ICADVM18.1-64b - Log: /tmp/rishu.log
File Tools Options Help

COPYRIGHT (C) 1992-2019 CADENCE DESIGN SYSTEMS INC. ALL RIGHTS RESERVED.
(C) 1992-2019 UNIX SYSTEMS Laboratories INC.,
Reproduced with permission.

This Cadence Design Systems program and online documentation are
proprietary/confidential information and may be disclosed/used only
as authorized in a license agreement controlling such use and disclosure.

RESTRICTED RIGHTS NOTICE (SHORT FORM)
Use/reproduction/disclosure is subject to restriction
set forth at FAR 1252.227-19 or its equivalent.
Program: @(#)CDS: virtuoso version ICADVM18.1-64b 10/08/2019 19:07 (cpgbld02.cadence.co
Sub version: sub-version ICADVM18.1-64b.500.7 (64-bit addresses)
Initializing Virtuoso Multi Technology Framework.
Initializing Virtuoso Photonics (VPO).
Loading geView.cxt
Loading menuBuilder.cxt
Loading schView.cxt
Loading selectSv.cxt
Loading wireEdit.cxt
Loading pte2.cxt
Loading xLUI.cxt
Loading auCore.cxt
Loading vhd1.cxt
Loading seismic.cxt
Loading ci.cxt
Loading layers.cxt
Loading ams.cxt
Loading cli.cxt
Loading cle.cxt
Virtuoso Framework License (111) was checked out successfully. Total checkout time was 0.39s.
Loading oasis.cxt
Loading analog.cxt
```

Related Information

[Licensing in Design Framework II](#)

[Using the Command Interpreter Window](#)

Generating Layout

In Virtuoso, you can use the Virtuoso Suite EXL layout editor (Layout EXL) to generate a single layout that allows you to define an initial placement of the electrical and optical components in the design.

This sections covers the following topics.

- Opening the Library Manager
- Creating a New Library
- Creating a New Library Cellview
- Creating a Schematic Cellview
- Creating a Schematic Cellview
- Creating a Layout Cellview
- Specifying the Components to be Generated
- Specifying the I/O Pins to be Generated
- Defining the Optical Pin Attributes

Opening the Library Manager

All elements of a Virtuoso design—symbols, schematics, and layouts—belong to a library. Virtuoso can support several libraries simultaneously and a Virtuoso designer often works in a design library, which references a given fabrication process and uses the elements recommended for that process. Such a library is called a *process* library and it contains technology definitions in addition to containing information about basic devices. Optionally, a designer can use additional design elements from another library, called the *reference* library.

Virtuoso also supports some generic, process-independent libraries, such as `basic` and `analogLib`, which include common design elements.

Note: The generic libraries do not contain layout implementations for any elements.

To access the various libraries, you can open the Library Manager form in stand-alone mode from an `xterm` or a command line window or within the Virtuoso design environment using the CIW.

To open the Library Manager in standalone mode:

- ➔ In a terminal window, type the following command:

```
libManager &
```

The Library Manager appears as a stand-alone application, which is not integrated in the Virtuoso design environment.

Virtuoso Photonics Solution Guide

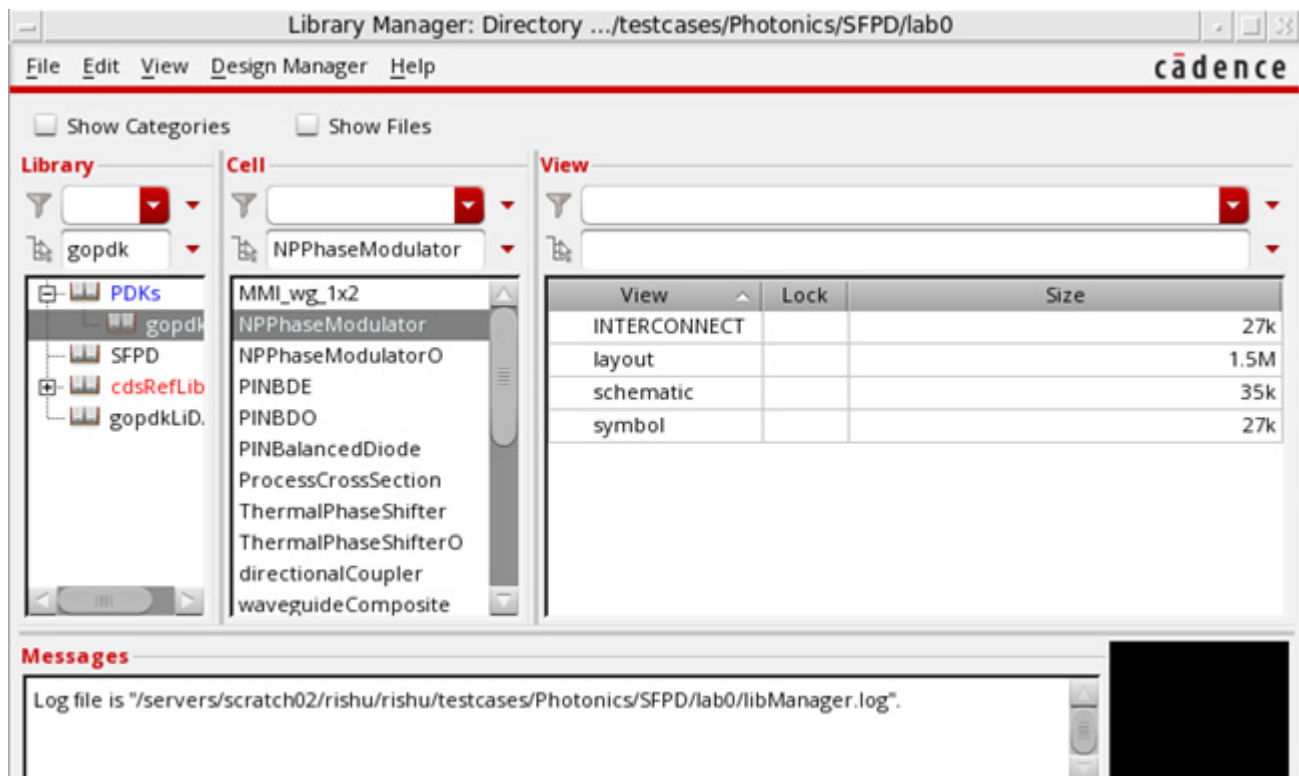
Using EPDA in the Virtuoso Environment

Note: In stand-alone mode, you cannot open cellviews.

To open the Library Manager in Virtuoso Design Environment:

➔ In CIW, choose *Tools – Library Manager*.

The Library Manager form is displayed in the Virtuoso design environment as displayed in the figure.



Related Information

[Library Manager Form](#)

Creating a New Library

In Virtuoso, you can create a new library using one of the following methods:

- [Creating a New Library using the Library Manager](#)
- [Creating a New Library Using the CIW](#)

Creating a New Library using the Library Manager

To create a new library using the Library Manager:

1. In the Library Manager, choose *File – New – Library*.

Alternatively, you can click inside the *Library* list box and press `Ctrl+N` on the keyboard.

You can also type the name of the library in the *Library* field and press `Ctrl+N` to open the New Library form. In this case, the *Name* field in the New Library form is automatically populated with the name that you have entered in the *Library* field.

The New Library Form is displayed.

Note: Creating a new or temporary library within an existing library is not allowed because any directories within a library are handled as cells.

2. In the *Name* field, type the name of the library you want to create.

The new library name cannot be the same as another library.

3. The *Directory* navigation tools (list boxes and toolbar buttons) to specify the destination directory in which you want to create the new library. You can also type a directory path in the *Directory* field.

You must have permission to edit the directory in which you want to create a library.

Note: If you want the library to be under design management control, you must create it in a managed project area. For additional information about creating managed libraries, see the [Virtuoso Software Licensing and Configuration Guide](#).

4. (Optional) In the *Design Manager* group box, specify whether you want to use a design management system.

- ☐ If you want to use a design management system, select *Use <design management system>* (the default).
- ☐ If you do not want to use a design management system, select *Use No DM*.

Note: If there is no design management system available, *No design manager setup found* will be displayed.

For more information about design management systems, see [Design Manager](#).

5. You can select the *Compression Enabled* check box to write OpenAccess data to this library in a compressed format. For more information, see [Compressing a Library Using Library Manager](#).

6. Click *OK*.

The Technology File for New Library form is displayed.

7. Choose the *Reference existing technology libraries* technology file options.

For more information about the various technology files options, see [Technology File for New Library Form](#).

For more information about technology files, see the [Virtuoso Technology Data User Guide](#).

Creating a New Library Using the CIW

To create a new library using the CIW:

1. From the CIW, select *File – New – Library*.

The New Library form is displayed.

Note: Creating a new or temporary library within an existing library is not allowed because any directories within a library are handled as cells.

2. Enter a *Name* for your new library.
3. (Optional) Choose the name of the *Directory (non-library directories)* that you want to store your library in. Otherwise, you can specify the library path in the text box under the *Directory (non-library directories)* section.

Note: By default, the library will be created and stored in the current directory.

4. Choose the *Reference existing technology libraries* technology file options.

For more information about the various technology files options, see [Technology File for New Library Form](#).

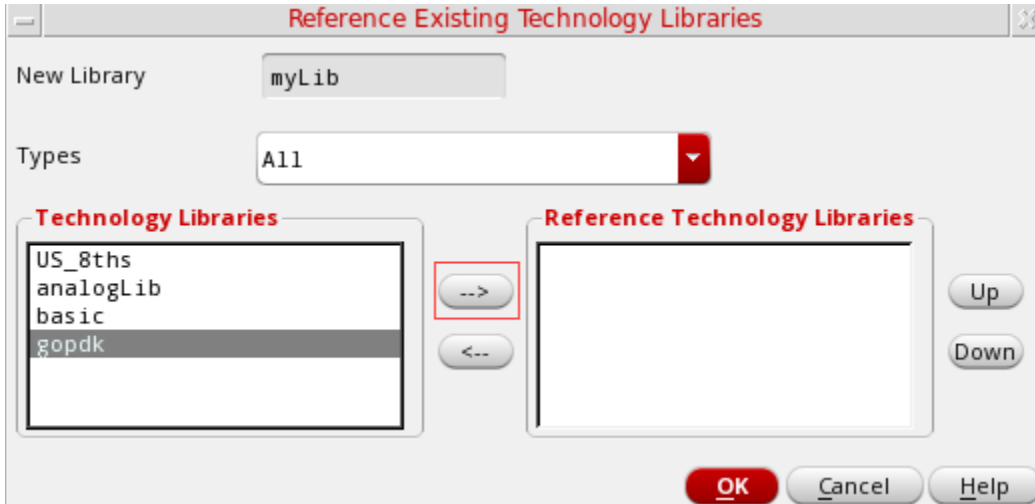
For more information about technology files, see the [Virtuoso Technology Data User Guide](#).

5. In the *Reference Existing Technology Libraries* form, double-click the *gopdk* library in the left column to move it to the right column.

Virtuoso Photonics Solution Guide

Using EPDA in the Virtuoso Environment

Alternatively, you can move items from one column to the other by selecting the item and then clicking the appropriate arrow key to specify the direction of move.



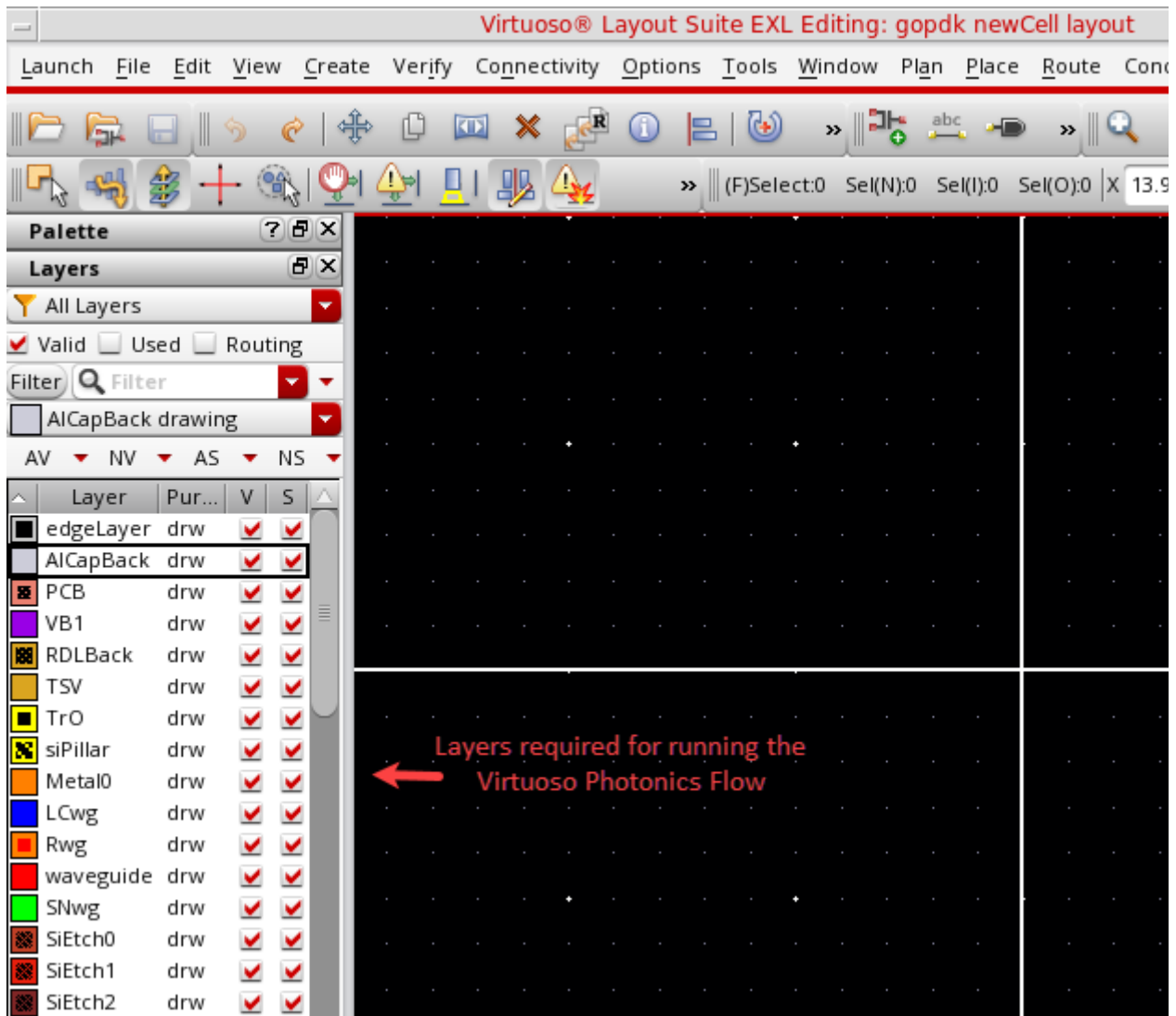
The new library is created.

6. Create a new layout cellview to verify if you have referenced the appropriate technology. See [Creating a Layout Cellview](#).
7. In the layout cellview opened in Layout EXL, right-click the menu bar and choose *Assistants – Palette*.

Virtuoso Photonics Solution Guide

Using EPDA in the Virtuoso Environment

8. The layout cellview displays the Palette assistant, as shown in the figure below. You can use the Palette assistant to verify that the layers required for running the Virtuoso Photonics Flow are supported.



Related Information

Palette Assistant

Creating a New Library Cellview

To create a new cellview:

1. Open the Library Manager.

Note: Alternatively, you can also create a new cellview from the CIW by following the same instructions.

2. Choose *File – New – Cell View*.

Alternatively, you can click inside the *Cell* or *View* list box and press `Ctrl+N` on the keyboard.

The New File form is displayed.

You can also type the name of the cell in the *Cell* field and press `Ctrl+N` to open the New File form. In this case, the *Cell* field in the New File form is automatically populated with the name that you have entered in the *Cell* field of the Library Manager form.

3. In the *Library* drop-down list, choose the name of the library in which you want to create a new cellview.
4. In the *Cell* field, type a cell name for the new cellview.
5. In the *View* field, type a view name for the new cellview.
6. In the *Type* drop-down list, choose the type of view to be opened.
7. In the *Application* section, select *Layout EXL*.
8. (Optional) Select *Always use this application for this type of file* to always open the selected application when the selected view type is opened.
9. Click *OK*.

The new cellview opens in the selected application.

Related Information

[Creating a Layout Cellview](#)

Creating a Schematic Cellview

In Virtuoso, the schematic representation of the design forms the starting point for a design. As in the electrical flow, the schematic is also used to launch the circuit simulation and to generate a layout in the Virtuoso Photonics Flow. You can also update the schematic based on any design decisions made during the simulation or layout generation. After the layout updates have been backannotated to the schematic, you can use the updated schematic to run any verification checks. Therefore, it is important to create a good schematic design that

can serve all these purposes and help generate a connectivity-driven, correct-by-construction layout. For more information on creating schematics in Virtuoso, see the [Virtuoso Schematic Editor User Guide](#).

Note: Virtuoso supports the implementation of photonic Pcells in the same manner as that of electrical ones. Therefore, when you are in the EPDA framework, most of the Virtuoso features will continue to work the same way as in the pure electrical flow.

To launch the Virtuoso Schematic Editor to create a new schematic cellview:

1. Open the Library Manager.

Note: Alternatively, you can create a new cellview from the CIW by following the same instructions.

2. Choose *File – New – Cell View*.

Alternatively, you can click inside the *Cell* or *View* list box and press `Ctrl+N` on the keyboard.

The New File form is displayed.

3. In the *Library* drop-down list, choose the name of the library in which you want to create a new cellview.

If you have created any new libraries, they should be available in the list.

4. In the *Cell* field, type a cell name for the new cellview.

5. In the *View* field, type *schematic* to open the cell in the schematic view.

6. In the *Type* drop-down list, choose *schematic*.

7. In the *Application* section, choose *Schematics XL*.

8. Select *Always use this application for this type of file* to always open the selected application when the selected view type is opened.

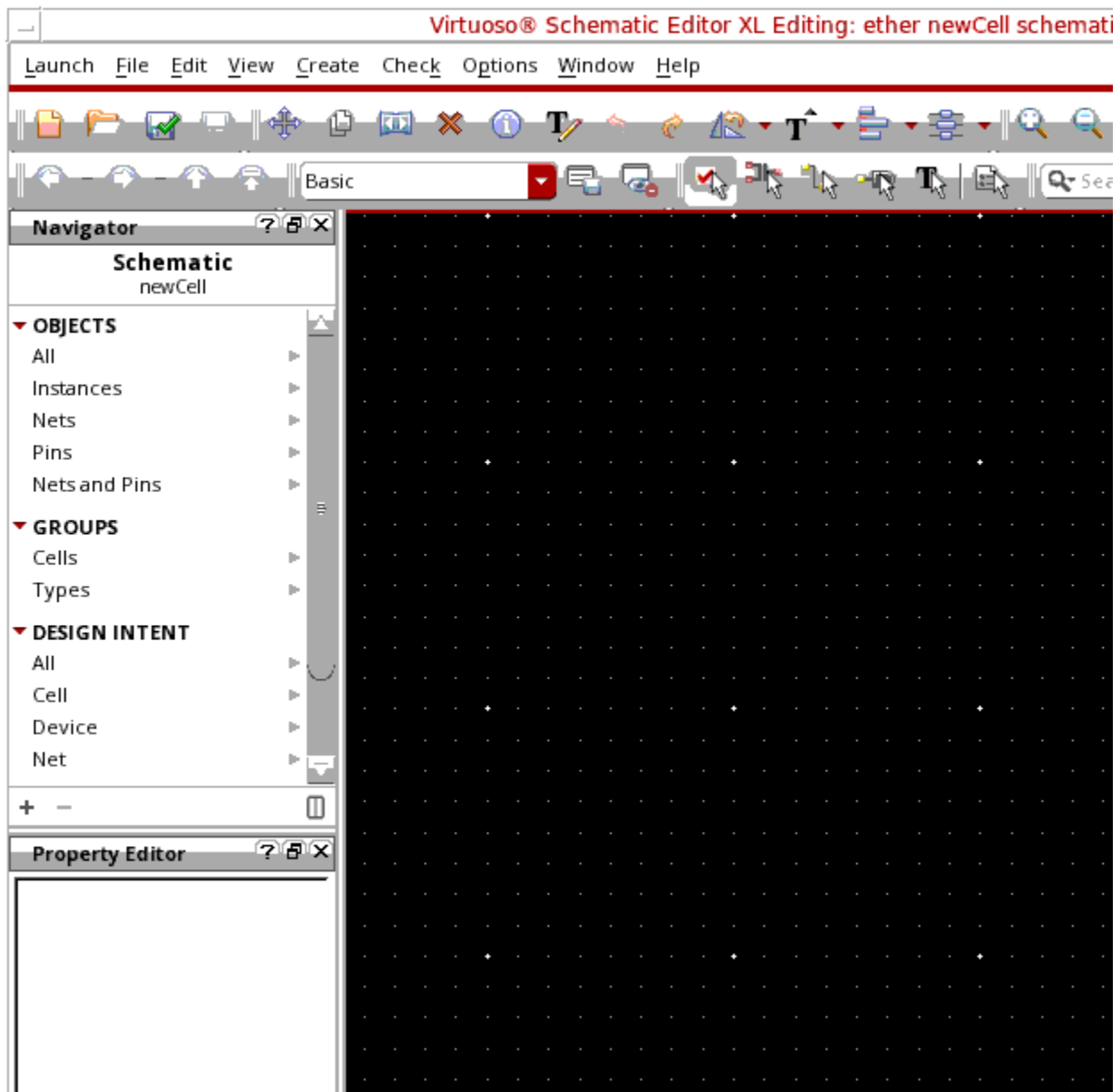
9. Choose the appropriate *Open for* option.

10. Click *OK*.

Virtuoso Photonics Solution Guide

Using EPDA in the Virtuoso Environment

The new cellview opens in the selected application, as shown in the figure below.



Related Information

Creating Schematics

Opening a Schematic Cellview

To open an existing schematic cellview:

1. Open the Library Manager.

Note: You can also open a new cellview from the CIW by following the same instructions.

2. Choose *File – Open – Cell View*. Alternatively, you can click inside the *Cell* or *View* list box and press `Ctrl+N` on the keyboard.

The Open File form is displayed.

3. In the *Library* drop-down list, choose the name of the library from which you want to open the schematic cellview.

If you have created any new libraries, they should be available in the list.

4. In the *Cell* field, type the name of the cell to open.
5. In the *View* drop-down list, choose *schematic*.
6. In the *Application* section, choose *Schematics L*.
7. Click *OK*.

The schematic cellview opens in the Virtuoso Schematic Editor window. If the schematic you opened includes both electrical and optical connections, the schematic editor displays the connections differently.

Related Information

Creating Schematics

Creating a Layout Cellview

To create a layout cellview:

1. Open the Library Manager.

Note: Alternatively, you can create a new cellview from the CIW by following the same instructions.

2. Choose *File – New – Cell View*.

Alternatively, you can click inside the *Cell* or *View* list box and press `Ctrl+N` on the keyboard.

The New File form is displayed.

3. In the *Library* drop-down list, choose the name of the library in which you want to create a new cellview.

If you have created any new libraries, they should be available in the list.

4. In the *Cell* field, type a cell name for the new cellview.
5. In the *View* field, type *layout* to open the cell in the layout view.
6. In the *Type* drop-down list, choose *layout*.
7. In the *Application* section, choose *Layout EXL*.

Note: The Virtuoso Photonics Flow is supported only in the Layout EXL application.

8. Select *Always use this application for this type of file* to always open the selected application when the selected view type is opened.
9. Choose the appropriate *Open for* option.
10. Click *OK*.

The new cellview opens in the selected application.

Specifying the Components to be Generated

Use the Generate All From Source toolbar button to generate layout representations of the schematic design components. Alternatively, you can choose the *Connectivity — Generate* menu command to open the Generate Layout form.

For the schematic instances that have layout representations available, the *Generate All From Source* command creates corresponding layout views in the canvas.

To specify the components to be generated:

1. In the Generate Layout form, select the *Generate* tab.
2. In the *Generate* group box, select the *Instances*, *I/O Pins*, and *PR Boundary* options, as required.
 - a. To chain transistors, select the *Chain* check box.

For information on chaining optical pins, see Generating Optical Chains.
 - b. To stop Layout EXL from generating layout pins for global nets in the schematic, select *Except Global Pins*.

3. To preserve any user-defined binding of devices between the schematic and the layout, select *Preserve User-Defined Bindings*.
4. Click *OK*.

Related Information

Generating All Components From Source

Specifying the I/O Pins to be Generated

You specify the pins to be generated on the *I/O Pins* tab of the Generate Layout form.

For each pin listed, the form shows the parameters that will be used to generate its equivalent in the layout. You can remove or change the specification of any of the listed pins, or add new pins to be generated.

When generating pins, Layout EXL uses the same naming convention as the Virtuoso Schematic Editor, allowing you to assign different names to terminals and nets. Where terminal and net names are different in the schematic, Layout EXL creates pins with the same terminal name as in the layout, even though the net name associated with the pin might be different.

Power and ground pins defined at a lower level of the design hierarchy are not listed on the *I/O Pins* tab but the pins are still generated in the layout view. Virtuoso issues a message in the CIW to notify you about the operation.

Specifying the Default Values for All Electrical Pins

To specify the default values for all electrical pins:

1. In the Generate Layout form, select the *I/O Pins* tab.
2. Select *Type* as *Electrical*.

This option is available if the schematic has both electrical and optical pin types defined.

2. Choose the default routing layer from the *Layer* cyclic field.

The list of pin layers is obtained from the technology information applied to the design.

3. Specify the default *Height* and *Width* of the electrical pins and the *Number* of pins you want to create.
1. Click *Apply*.

The specified values are applied to all the electrical pins displayed in the list box.

Related Information

Specifying the I/O Pins to be Generated

Defining the Optical Pin Attributes

When generating optical pins, Layout EXL uses the same naming convention as specified in the Virtuoso Schematic Editor, allowing you to assign different names to ports and nets. Where port and net names are different in the schematic, Layout EXL creates pins with the same terminal name as in the layout, even though the net name associated with the pin might be different.

If a pin is associated with an optical net, Layout EXL automatically generates optical pins on an optical layer, such as the `waveguide` layer. By default, the layer to be used for the optical pin generation is determined from the value set for the `phoPinLayer` environment variable. If the environment variable does not have a valid layer set, the first photonic layer in the `validLayer` list is used.

Each optical pin can have the following attributes defined:

- **Width:** Width of the waveguide associated with the photonic port.
- **Radius:** Curvature of the waveguide associated with the photonic port.
By default, the radius is not defined.
- **Angle:** Access direction of the photonic port.

Generating Optical Pins

To generate an optical pin in the layout:

1. In the Generate Layout form, select the I/O Pins tab.
2. Select *Type* as *Optical*.

This option is available if the schematic has both electrical and optical pin types defined.

3. Choose the default routing layer from the *Layer* cyclic field.

The default pin layer is determined from the value set for the `phoPinLayer` environment variable.

4. Specify the default *Width*, *Radius*, and *Input Angle* of the pins you want to create.

The default value set for *Input Angle* applies only to the input pins. The output pins are set to a complementary angle value.

5. Click *Apply* to generate the pins in the layout.

Updating Optical Pins

1. In the Generate Layout form, select the I/O Pins tab.
2. Select *Type* as *Optical*.

This option is available if the schematic has both electrical and optical pin types defined.

3. Specify the new *Layer*, *Width*, *Radius*, and *Angle*, as appropriate.
4. Click the *Update* button.

The selected pins are updated.

Generating Selected From Layout

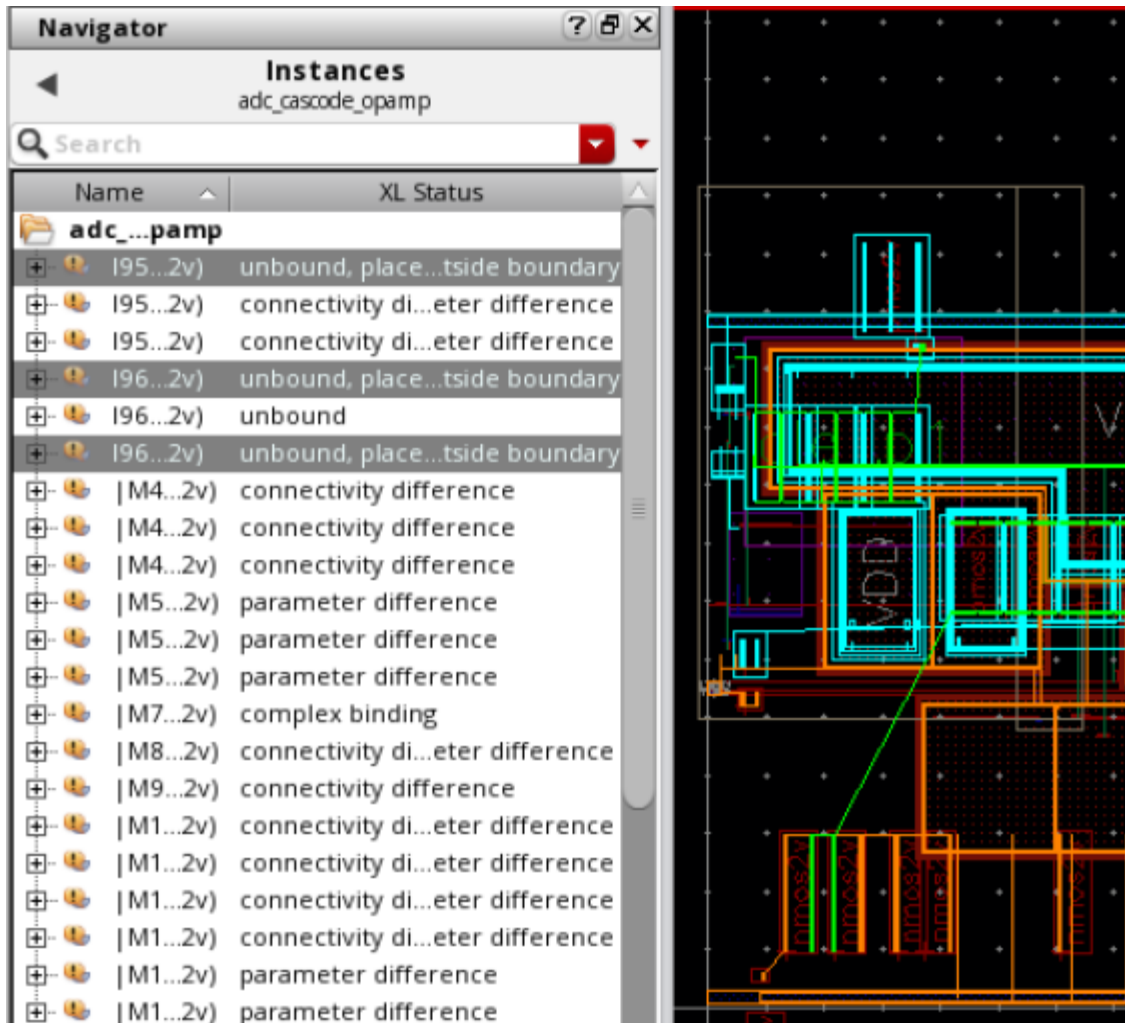
In Virtuoso Layout Suite EXL, if you have the `Virtuoso_Photonics_Option` license checked out, you can select unbound layout components and generate them in the schematic.

To generate the selected layout components in the corresponding schematic window:

Virtuoso Photonics Solution Guide

Using EPDA in the Virtuoso Environment

1. In the Layout EXL Navigator assistant, select the instances or terminals that display their *XL Status* as *unbound*.



2. Choose *Connectivity – Generate – Selected From Layout*.

Alternatively, click the *Generate Selected From Layout* button on the Layout XL toolbar.

If the corresponding schematic cellview is read-only, a message is issued to prompt that the cellview be made editable.

If the corresponding schematic cellview is editable and the schematic has not already been extracted, a dialog box pops up to ask if the schematic cellview can be extracted.

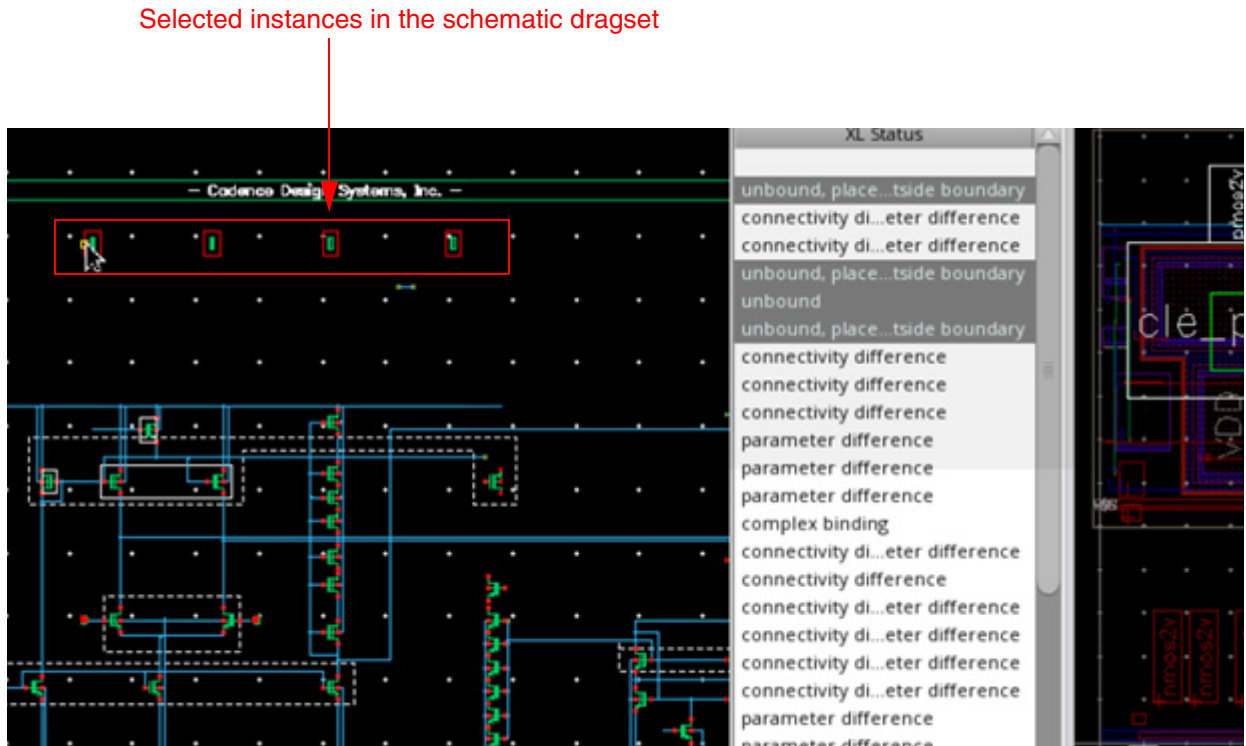
3. Press F3 to display the Generate Selected From Layout form to customize how dummies are placed in the schematic. (Optional).

Virtuoso Photonics Solution Guide

Using EPDA in the Virtuoso Environment

4. Click **OK** to proceed with the extraction of the schematic cellview.

The selected instances are added to the dragset and are available for placement in the schematic view.

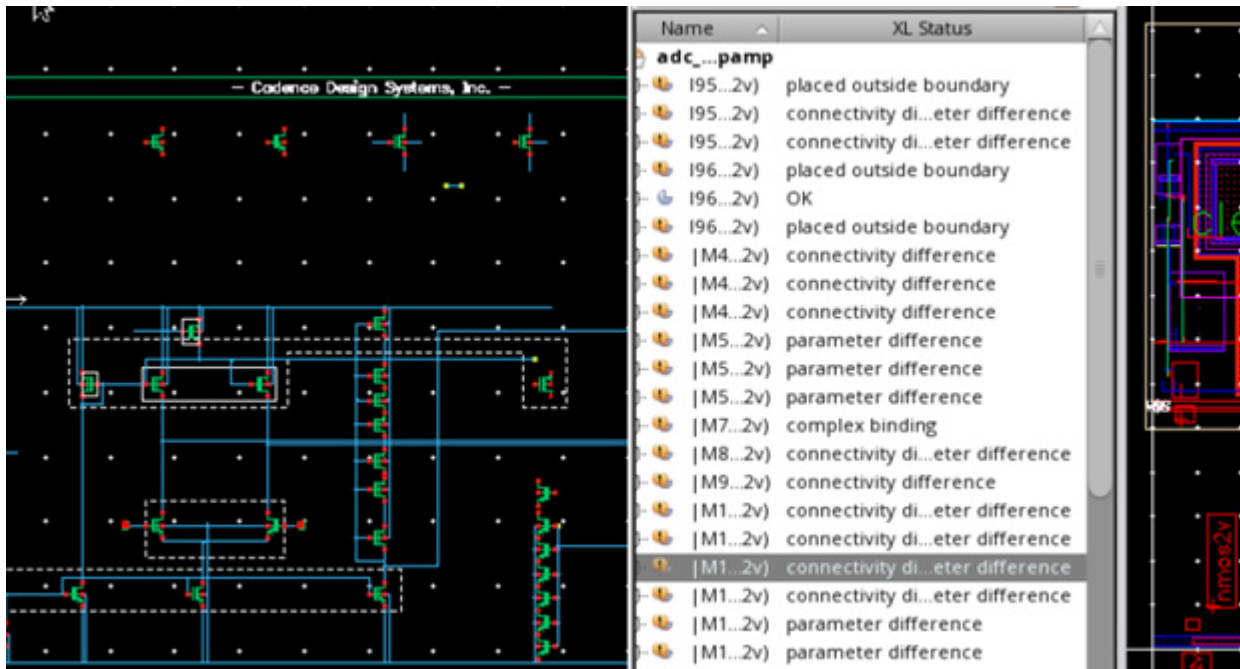


5. Click anywhere in the schematic canvas to place the selected instances.

Virtuoso Photonics Solution Guide

Using EPDA in the Virtuoso Environment

The selected instances are placed in a row in the schematic cellview, reflecting their layout connectivity. The *XL Status* of the instances in the layout view is updated to reflect the current status.



Note: If the instance or terminal created in the schematic has a master difference with the layout, you can update the physical binding in CPH to resolve the mismatch.

Related Information

[Generate Selected From Layout Form](#)

[IxHiGenerateSelectedFromLayout](#)

[Generating Layout](#)

Abutting Photonic Elements

The Virtuoso Photonics Flow supports all the same abutment features for photonic elements as are supported for electronic elements. In addition, the flow allows a type of abutment, which aligns the center point of an optical port with the matching facet information (*width*, *angle*, or *layer*). This enables any-angle abutment support for photonic elements.

For photonic elements, abutment is triggered by overlapping photonics ports, provided the corresponding routing layers in the technology database have been set up to trace optical connectivity.

For photonic layouts, abutment is also enabled for optical pins of non-Pcell devices. If you set the `phoAbutNonPcells` environment variable to `t`, Layout EXL also supports the abutment of non-Pcell instances with Pcell instances and other non-Pcell instances. During non-Pcell abutment, top-level pins can abut with instance pins. By default, instance pins on the same pin layer are abutted and aligned at the center.

For photonic designs, waveguide abutment is also supported. However, for waveguide abutment to be possible, the two abutting waveguides must be perfectly aligned and the abutting ports must be on the same net. If the abutting waveguides result in any shorts, the corresponding markers are generated in the [Annotation Browser](#) assistant.

To workaroud the following situations that could otherwise prevent abutment, you can set the `phoAbutAutoAdjust` environment variable to enable abutment.

- If the optical port is locked and the overlapping instance is at a different angle, setting the `phoAbutAutoAdjust` environment variable automatically moves and adjusts the instance angle to be compatible with the port to allow abutment. If the port is unlocked and moved towards the instance, the angle of the optical port automatically adjusts to be compatible with the instance to allow abutment.
- If the overlapping optical port differs from the abutting instance in width, setting the `phoAbutAutoAdjust` environment variable automatically adjusts the width of the port to enable abutment.

Photonics devices can be spaced out in case of abutment failure when the spacing properties `vxllInstSpacingDir` and `vxllInstSpacingRule` are set correctly on the overlapping pin figures.

Related Information

[Device Abutment](#)

Generating Optical Chains

In the EPDA framework, you can use the following Layout XL commands to chain optical waveguide instances:

- *Generate All From Source*
- *Update Components And Nets*

- *Generate Selected From Source*
- *Generate Chained Devices*

When using *Generate All From Source* and *Update Components And Nets*, the layout instances selected for chaining are checked using the `phoIsWaveguide` SKILL function to verify them as waveguide instances. If the selected instances are found to be waveguides, they are chained during layout generation.

However, when layout generation is being performed using the *Generate Selected From Source* command, all the connected optical devices are chained. The `phoIsWaveguide` SKILL function is not used in this case.

You can also chain waveguide instances and top-level photonic pins using the Virtuoso® Layout Suite XL *Connectivity – Generate – Chained Devices* command or use the context-sensitive *Chain* menu option. When the selected set includes only waveguide instances and photonic pins, the *Chain* context-sensitive menu has only the *Default* option enabled. The other menu options, *Top*, *Center*, and *Bottom*, are disabled in this case.

Note: The following interactive chaining options are not supported for photonic devices:

- *Preserve Existing Chains*
- *Use Device Order*
- *Interdigitate Chains*
- *Mirror*
- *Permute Pins*
- *Align PMOS*
- *Align NMOS*

Related Information

[1xChain](#)

[Manual Device Abutment](#)

[Create Pin Options: Virtuoso Photonics Option](#)

[Composite Waveguide Editor Form](#)

[Abutment in Virtuoso Photonics Solution](#)

Generating an Incremental Chain

When using the Virtuoso Photonics Solution, you can incrementally create a chain by adding an object to an anchor. An anchor can be an instance or a top-level pin object, or a group of such abutted objects, which retains its position after chaining.

When *Incremental* chaining is invoked on an anchor, Virtuoso prompts to select an appropriate object for chaining by highlighting the suitable chaining connections on the nets associated with the selected anchor. Select any of the objects connected to the highlighted connections. You can iteratively add instances to the chain until the *Incremental* chaining command is interrupted by pressing the `ESC` key.

To incrementally chain instances:

1. In the layout canvas or the Navigator assistant, right-click an instance or a group of abutted objects to be used as an anchor.

2. From the shortcut command, choose *Chain – Incremental*.

All the candidate instances for chaining are highlighted in the layout canvas using probes.

3. Select an object to be added to the chain.

The selected object is added to the chain, with the anchor retaining its position.

4. To add another object to the chain, select the object. Do this iteratively, if more objects must be added to the chain.

5. To stop adding more objects to the chain, press the *Esc* key.

Related Information

[lxHiIChain](#)

Generating an Anchored Chain

When using the Virtuoso Photonics Solution, you can automatically chain instances by adding to a selected set of fixed anchors, using connectivity from the selected set. Anchor chaining allows chains to be created much faster compared to incremental chaining, which requires manually selecting each instance to be added to the chain.

To automatically create an anchored chain:

1. In the layout canvas or the Navigator assistant, right-click an instance or a group of instances to be used as an anchor.
2. From the shortcut command, choose *Chain – Anchor*.

All the candidate instances for chaining are highlighted in the layout canvas using probes.

Editing Layout

Virtuoso Layout Suite EXL provides a lot of editing capabilities to work with electro-phonic layout designs. You can use the layout editor to create custom layouts or edit any existing layout. From supporting basic parameter updates to layout components to enabling editing of composite waveguides, the Virtuoso Layout Suite EXL editor is well-equipped to support the editing of a Virtuoso-enabled Photonic design.

Editing the Layout Parameters

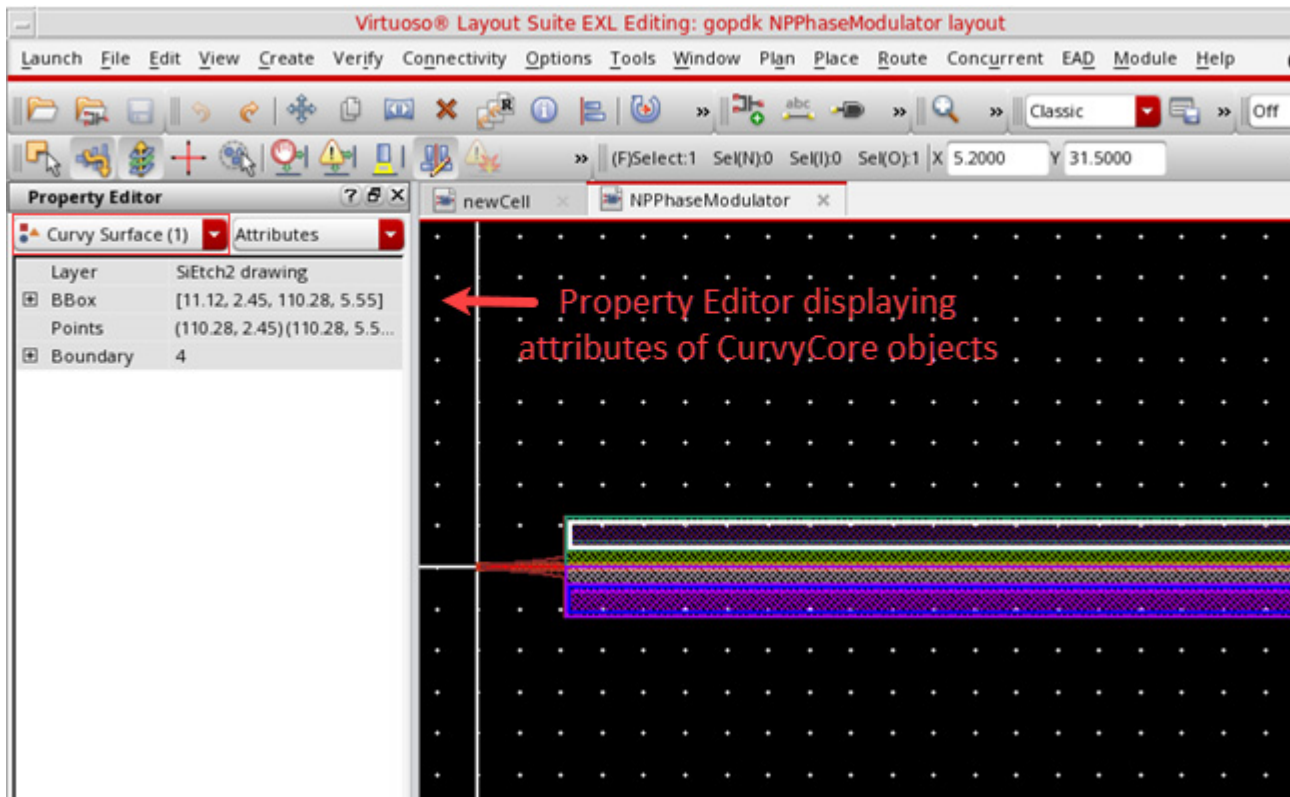
Photonics elements are formed as a result of complex mathematical calculations used to generate shapes. Therefore, these elements are “wrapped” into a generator and can be controlled through parameters. In fact, all the elements, including interconnect elements such as waveguides in a Photonic Integrated Circuit are defined as fixed or Pcell instances that can be controlled using parameters. Electronic circuit designs in Virtuoso, on the other hand, use a different methodology where the interconnects (wires) are often defined as top-level shapes. Because most photonic elements in a Photonic Integrated Circuit can be controlled through parameters, these instances can be interactively controlled using the Property Editor assistant or the Instance Property form.

The *Property Editor* assistant enables you to view and edit object property values for all electrical and optical components in your design. By default, each property is displayed in a separate table row in the *Property Editor*.

Virtuoso Photonics Solution Guide

Using EPDA in the Virtuoso Environment

The Property Editor table in the figure displays the properties for the selected CurvyCore® objects in the design.



For detailed information about the *Property Editor* user interface, how to work with it and how to edit property values, see [The Property Editor Assistant](#) in the *Virtuoso Schematic Editor User Guide*.

Editing the Composite Waveguides

In the Virtuoso Photonics Solution, Layout EXL enables you to use optical connections called waveguides in your design. When several such optical connections or waveguides are used in a design, the resultant waveguide is called a composite waveguide.

The Virtuoso Photonics Solution supports the generation of such composite waveguides using the [Generate All From Source](#) command and supports editing of the composite waveguides using the Composite Waveguide Editor. Each component of the composite waveguide is called an *Element*.

Note: The Composite Waveguide Editor can be launched from both the editors, schematic and layout. If launched in the Schematic view, the Composite Waveguide Editor allows early estimation of the waveguide geometry.

The various ways you can launch the Composite Waveguide Editor in the schematic and layout view are:

- *From the Create Instance form*

When creating a composite waveguide instance, click the *Ellipses (...)* button corresponding to the *Edit* label displayed just above the *Waveguide* parameter list.

- *From the Property Editor assistant*

In the Property Editor assistant, click the *Ellipses (...)* button corresponding to the *Edit* label displayed just above the *Waveguide* parameter list.

- *From the Property Editor form*

In the *Parameters* tab of the Property Editor form, click the *Ellipses (...)* button corresponding to the *Edit* label displayed just above the *Waveguide* parameter list.

Editing a waveguide can involve the following:

- Adding a Waveguide Element
- Deleting a Waveguide Element
- Changing the Position of a Waveguide Element
- Adding an Optical Waveguide Connector
- Rotating a Waveguide Element

Adding a Waveguide Element

To add a waveguide element:

1. In the layout canvas, select the composite waveguide instance to update.

The Property Editor assistant populates to display the properties of the selected optical instance.

2. In the Property Editor assistant, click the *Ellipses (...)* button corresponding to the *Edit* label displayed just above the *Waveguide* parameter list.

The Composite Waveguide Editor displays.

3. In the Composite Waveguide Editor, right-click an existing waveguide element to use as the reference and choose *Add Before* or *Add After* to specify the position of the element to add.

The Add Element form displays.

4. Specify the *Library*, *Cell*, and *View* to use for the new waveguide element.
5. Click *OK* to apply the settings and close the editor.

A new waveguide element gets added at the position you specify.

Deleting a Waveguide Element

To delete a waveguide element:

1. In the layout canvas, select the composite waveguide instance to update.

The Property Editor assistant populates to display the properties of the selected optical instance.
2. In the Property Editor assistant, click the *Ellipses (...)* button corresponding to the *Edit* label displayed just above the Waveguide parameter list.

The Composite Waveguide Editor displays, listing the various waveguide elements that comprise the composite waveguide.
3. In the Composite Waveguide Editor, right-click the waveguide element that you want to remove and choose *Delete* from the context-sensitive menu.

The selected element is removed from the waveguide element list and from the layout canvas.
4. Click *OK* to apply the settings and close the editor.

Changing the Position of a Waveguide Element

To change the position of an element in the composite waveguide:

1. In the layout canvas, select the composite waveguide instance to update.

The Property Editor assistant populates to display the properties of the selected optical instance.
2. In the Property Editor assistant, click the *Ellipses (...)* button corresponding to the *Edit* label displayed just above the *Waveguide* parameter list.

The Composite Waveguide Editor displays, listing the various waveguide elements that comprise the composite waveguide.
3. In the Composite Waveguide Editor, right-click the waveguide element for which the position needs to be changed and choose *Move Up* or *Move Down* to specify the new position for the element.

The selected element is moved to the new position in the element list. The composite waveguide displayed in the layout canvas also reflects the element at its new position.

Adding an Optical Waveguide Connector

Optical waveguide connectors are complex waveguides that can automatically generate a waveguide layout by using basic input information such as curve type.

To add an optical connector:

1. In the layout canvas, select the composite waveguide instances to connect.
2. Right-click and choose *Optical Connector* from the context-sensitive instance menu.

The Waveguide Connector for Optical Connections dialog box opens.

3. From the *Connector* drop-down list, choose a connector type to use for connecting the selected waveguides.

Note: The second value in the listed *Connector* options indicates the style of the connector that will be created—*curve*, *sine*, and so on.

4. Click *OK*.

An optical connector is created in the layout canvas based on the connector style you selected.

Rotating a Waveguide Element

If a composite waveguide has the rotation parameter set, you can rotate all the elements inside the composite waveguide by the specified rotation value. The rotation of the waveguide elements is determined based on the rotation value set on the individual waveguide elements and the value set on the composite waveguide.

Let us consider that a composite waveguide includes two `waveguideStraight` elements with rotation on each element set to 0. Let us also assume that the rotation of the composite waveguide is set to 45.

On canvas, both the waveguide elements are rotated by 45 degrees, which puts them at an effective rotation of 45 degrees. If the `waveguideStraight` elements were set to rotation of 45 and the composite waveguide was set to rotate by 45 degrees, the effective, on-canvas rotation of the `waveguideStraight` elements will be 90 degrees.

However, when you edit a composite waveguide using the Composite Waveguide Editor, the composite waveguide is considered to be at rotation 0. This allows editing the rotation of the

individual waveguide elements without taking into account the rotation of the composite waveguide.

For more information on rotating objects using the Virtuoso Photonics Solution, see [Rotating Objects: Virtuoso Photonics Solution](#) in the *Virtuoso Layout Suite XL: Basic Editing User Guide*.

Related Information

[Composite Waveguide Editor Form](#)

[Generating Optical Pins](#)

Managing the Layout Constraints

Use the **Constraint Manager** assistant to add, modify, check or delete constraints in your design.

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 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](#).

Routing Layout

To complete the electrical routing, Virtuoso supports the use of assisted routing for Virtuoso Photonics Solution.

To run the assisted router, you can use the options available in the *Virtuoso Space-based Router* toolbar.



Notice the *Lidar* icon in the *Virtuoso Space-based Router* toolbar, which has been introduced to set the required Wire Editing capabilities for Photonic Integrated Circuits. Virtuoso supports automatic routing for photonic designs, following the same rules as defined for the electrical layers.

For more information on how to use the various advanced routing capabilities of the *Virtuoso Space-based Router* to plan an efficient routing run for the Photonic Integrated Circuit, see the [Virtuoso Space-based Router User Guide](#).

Verifying Design

Checking a Layout Against a Schematic

To check the components in your layout view against the schematic view:

1. From the layout window menu bar, choose *Connectivity – Check – Against Source*.

Note: The *Check Against Source* command is also available through the *Check Against Source* icon () in the Layout XL toolbar.

The Check Against Source form is displayed.

2. In the *Check* group box, choose the differences you want to report.
3. In the *Output* group box, choose whether you want to open the CAS workspace or the Info window, or both, and specify whether you want to overwrite the log file from a previous CAS run, or append the results to the log file.
4. Specify the name of the log file in which the report of the CAS run can be logged.
5. Click *OK* to run the check.

The schematic is extracted and the CAS report is generated and displayed based on the *Output* options that you selected.

- ☐ If *Open workspace* is selected, Layout XL opens the CAS workspace to display the generated markers in the Annotation Browser CAS tab.

- ❑ If *Display info window* is selected, the schematic versus layout differences for the selected checks are reported in an information window.

6. In the Info window, choose *File – Save As to* save the report to an ASCII file.

Related Information

Check Against Source

Checking XL Compliance

To check if your design fulfills the compatibility criteria that allow it to fully leverage the connectivity-driven features of Virtuoso Layout Suite EXL, run the *Connectivity – Check – XL Compliance* command.

Note: Alternatively, you can call the [IxCheck](#) SKILL function to verify XL-compliance of a layout. To check each unique layout master in the layout hierarchy, use the [IxHierCheck](#) SKILL function.

The XL Compliance check evaluates the design for device correspondence with the schematic and reports information about ungenerated and unbound devices, if any. The report is intended to help you resolve any XL-compliance issues beforehand so that you can take full advantage of the numerous connectivity-driven capabilities provided by Layout EXL for optimal layout generation.

Note: The XL Compliance report is issued in CIW and you can choose to also display the report in HTML, if the [xlComplianceHtmlOpen](#) environment variable is set in accordance.

Related Information

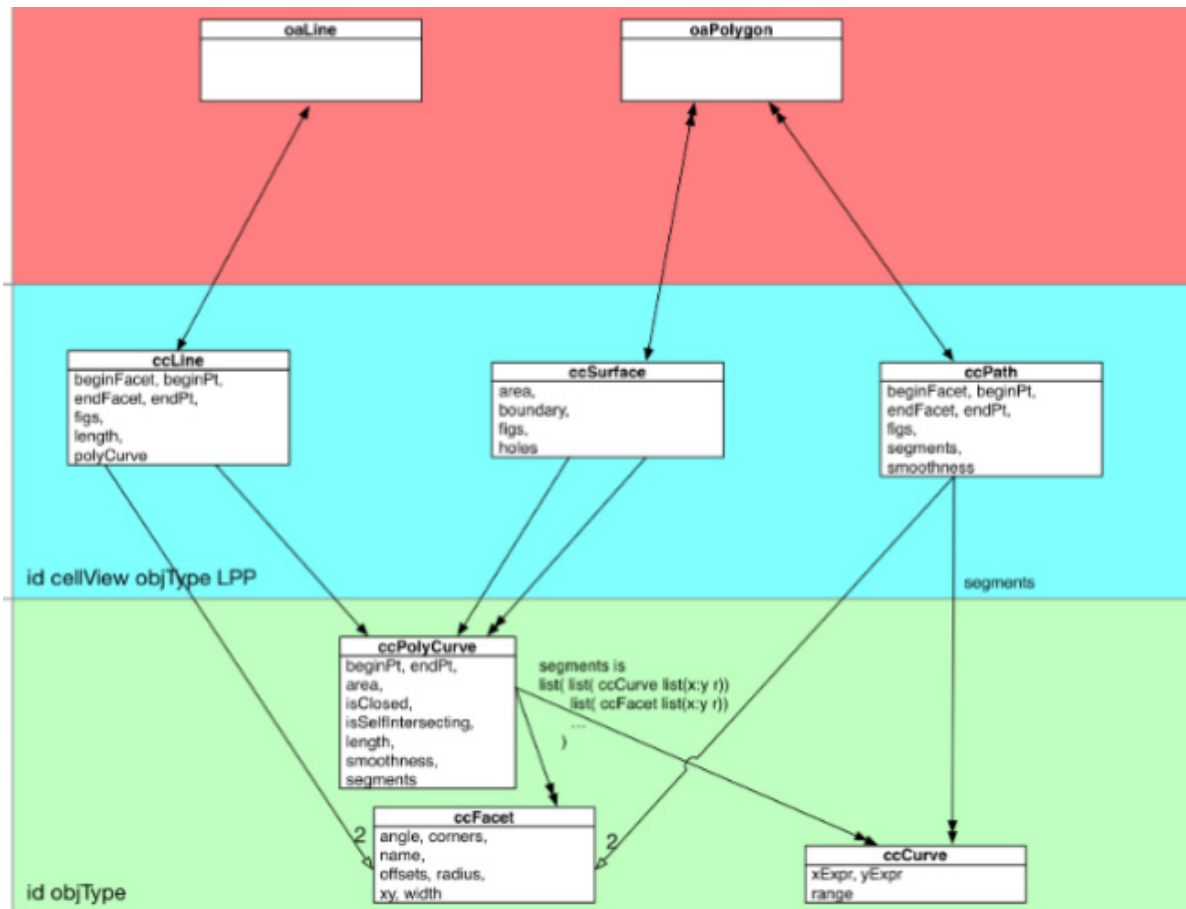
Checking XL Compliance

Virtuoso Photonics Solution Guide

Using EPDA in the Virtuoso Environment

CurvyCore Building Blocks

The CurvyCore data model is made up of three layers, the base layer, which has a mathematical core, the layer-purpose pair (LPP) aware layer, and the physical layer.



The base layer comprises three mathematical objects, `ccCurve`, `ccFacet`, and `ccPolyCurve`.

Objects in LPP layer are aware of the relevant grid, but are not limited in the number of vertexes. As a set of objects, they are `ccFigs`. When a `ccFig` is discretized (or when geometry for `ccFig` is generated), facets are subjected to a special treatment to reflect the fact that they

Virtuoso Photonics Solution Guide

CurvyCore Building Blocks

represent interface vertexes between elements. Origin of each facet is automatically snapped to the closest point on the grid while angle of the facet is kept unchanged.

The physical layer or the OpenAccess level represents the manufactured polygons. These physical shapes have limitations in the number of vertexes. When a physical representation is generated for a `ccFig`, the algorithm takes into account the vertex number limitation and slices the figures appropriately, resulting in one (`ccFig`) to many (`oaFigs`) relationship.

There is a collection of `pho` and `cc` SKILL functions, above the base `cc` SKILL functions that provide higher-level functionality, such as a full-feature straight generator, including the interface for tapering, controlling modal property computation.

Note: As a PDK developer, you can either your elements build from the ground-up, getting full control and full visibility or quickly putting together a PDK using predefined elements. You can also use or refer the available generic photonics PDK `gopdk`.

Photonics also has `db`-level SKILL functions that can be used to deal with the signal type for optical connection and the optical port definition. For more information regarding the `db`, `cc` and `pho` SKILL functions related to Photonics, see [Photonics Functions](#) in *Virtuoso Design Environment SKILL Reference*.

This chapter discusses the following:

■ CurvyCore Mathematical Objects

- Curves
- Paths and Facets
- Surfaces and Facets

■ CurvyCore Pcells

■ Waveguide Connectors

- Straight Connectors
- Bend Connectors
- Curve Connectors
- Waypoint Connectors

CurvyCore Mathematical Objects

This section discusses the mathematical base layer of the CurvyCore engine. The curvilinear shapes are represented in the curvilinear mathematical domain as parameterized curves. In mathematics, these curves are described by their Cartesian equation as:

$$f(x, y) = c$$

Here x and y are Cartesian coordinates and c is a constant.

A parameterized representation of a curve is the representation used in the mathematical domain of the CurvyCore engine. This means that the first step to create a curvy shape is to find its parametric representation.

According to implicit function theorem, if:

$$\frac{\partial f}{\partial y} \big|_{(x_0, y_0)} \neq 0$$

Then, a curve can be described around that point as $y = f(x)$.

For example, if:

$$y = 2x - 1 \rightarrow 2x - y = 1 \rightarrow f(x, y) = 2x - y, c = 1$$

$$y = x^2 \rightarrow y - x^2 = 0 \rightarrow f(x, y) = y - x^2, c = 0$$

A curve can also be represented as a path traced by a point. This means that if x and y are Cartesian coordinates describing the position of a point of a curve, the curve is represented as:

$$\gamma(t) = (x(t), y(t))$$

where, γ specifies the location of the point at time t . The parameterized curve is described as:

A parameterized curve in \mathbb{R}^n is a map $\gamma: (\alpha, \beta) \rightarrow \mathbb{R}^n$, for some α, β with $-\infty \leq \alpha < \beta \leq \infty$

the symbol (α, β) denotes an open interval

$$(\alpha, \beta) = \{t \in \mathbb{R} \mid \alpha < t < \beta\}$$

Related Information

[ccCreateCurve](#)

[ccCreatePath](#)

[ccCreatePolyCurve](#)

[ccCreateSurface](#)

[ccGenFigs](#)

[ccLayerOr](#)

[ccLayerAnd](#)

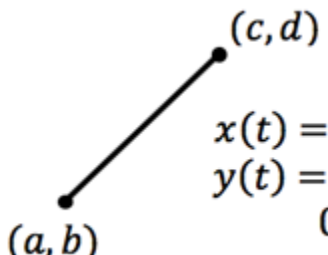
[ccSmoothenFig](#)

Curves

Curves are one of the primary data structures of a CurvyCore engine. Two main types of curves are `ccCurve` and `ccPolyCurve`.

ccCurve

`ccCurve` is one of the primary data structure of a CurvyCore engine. A `ccCurve` is a parametrized representation of a curve in a given interval. The following parametric equation is an example of the parametric equation that can be used to create a `ccCurve` representing a straight segment.


$$\begin{aligned}x(t) &= a + (c - a) * t \\y(t) &= b + (d - b) * t \\0 &\leq t \leq 1\end{aligned}$$

You can use the [ccCreateCurve](#) SKILL function to create a `ccCurve` object using its parametric representation. It retrieves the string representation of the Cartesian coordinates of the parameterized curve and the begin and end values of the parameter and returns a `ccCurve` object.

Virtuoso Photonics Solution Guide

CurvyCore Building Blocks

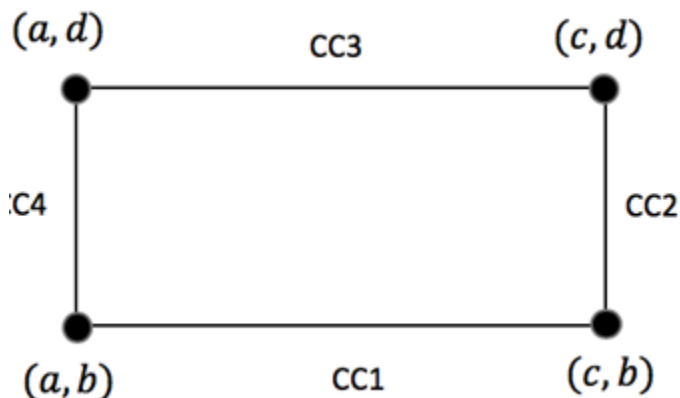
```
;ccCreateCurve(t_x t_y list(n_begin n_end))
;The following code creates a line segment between (0,0) and (2,4) and then
;displays the created ccCurve's attributes
LineSeg = ccCreateCurve("2*t" "4*t" 0:1)
cc00xb25fc433
LineSeg~>??
(cc00xb25fc433 objType "ccCurve" xExpr "(2*t)"
  yExpr "(4*t)" range
  (0.0 1.0)
)
```

The only parameter that can be used in the parametric representation of a curve is t , other parameters are numerical constants. This means that if you want to create a line segment whose endpoints need to be passed as a parameter, first you need to create the string representation using the numerical parameters and then create the `ccCurve` object.

ccPolyCurve

`ccPolyCurve` is a data structure that is used to stitch a sequence of `ccCurves` to create a more complicated curve. You can create a `ccPolyCurve` object using the [`ccCreatePolyCurve`](#) SKILL function.

To create a `ccPolyCurve` object from a rectangle shape, you need to consider that a rectangle is represented by line segments connecting its vertices.



Here the bounding box of a rectangle, `list(a:b c:d)`, generates `ccCurve` objects `CC1`, `CC2`, `CC3`, and `CC4` representing the edges of the rectangle. You can use the `ccCreatePolyCurve` SKILL function to join these edges.

Virtuoso Photonics Solution Guide

CurvyCore Building Blocks

The `ccCreatePolyCurve` SKILL function checks if the `polyCurve` object is a simple closed `polyCurve` and what is the level of smoothness between the segments while the `polyCurve` object is being created. For example, when a rectangle is generated from line segments, you might want to check whether it is continuous. `G0` enforces the continuity check and generates an error if the `polyCurve` object is not continuous.

In the following example, first the `CCSCreateLineSegment` function is defined to create and return a `ccCurve` object representing a line segment between two points.

```
procedure(CCSCreateLineSegment(point1 point2)
  let((ParameterizedLine XcoordPoint1 YcoordPoint1 XcoordPoint2 YcoordPoint2)
    XcoordPoint1 = xCoord(point1)
    YcoordPoint1 = yCoord(point1)
    XcoordPoint2 = xCoord(point2)
    YcoordPoint2 = yCoord(point2)
    ParameterizedLine.ParamX = lsprintf("%n + (%n-%n)*t", XcoordPoint1,
    XcoordPoint2,XcoordPoint1);X(t)=XcoordPoint1 + (XcoordPoint2-XcoordPoint1)*t
    ParameterizedLine.ParamY = lsprintf("%n + (%n-%n)*t", YcoordPoint1,
    YcoordPoint2,YcoordPoint1);X(t)=YcoordPoint1 + (YcoordPoint2-YcoordPoint1)*t
    ccCreateCurve(ParameterizedLine.ParamX ParameterizedLine.ParamY 0:1.)
  );let
);procedure
```

Next, the `CCSCreateRectangle` function is defined to use `CCSCreateLineSegment` to create and return a `polyCurve` object representing a rectangle with positive area as shown below.

```
procedure(CCSCreateRectangle(bBox)
  let((llx lly urx ury CC1 CC2 CC3 CC4)
    ury = yCoord(upperRight(bBox))
    urx = xCoord(upperRight(bBox))
    lly = yCoord(lowerLeft(bBox))
    llx = xCoord(lowerLeft(bBox))
    CC1 = CCSCreateLineSegment(llx:lly urx:lly)
    CC2 = CCSCreateLineSegment(urx:lly urx:ury)
    CC3 = CCSCreateLineSegment(urx:ury llx:ury)
    CC4 = CCSCreateLineSegment(llx:ury llx:lly)
    ccCreatePolyCurve(list(CC1 CC2 CC3 CC4)
      ?close t
      ?smoothness 'G0)
  );let
);procedure
```

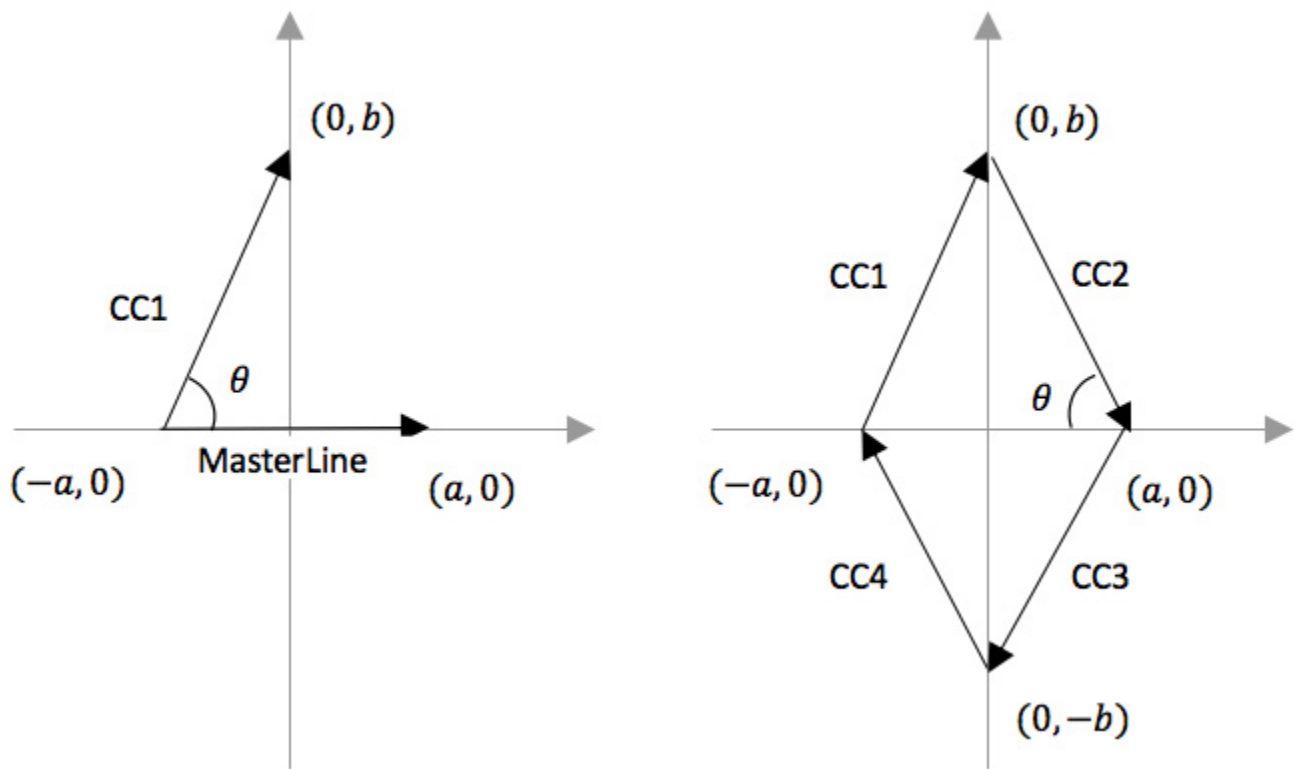
The following example shows how to check the `ccPolyCurve` attributes associated with the created rectangle.

Virtuoso Photonics Solution Guide

CurvyCore Building Blocks

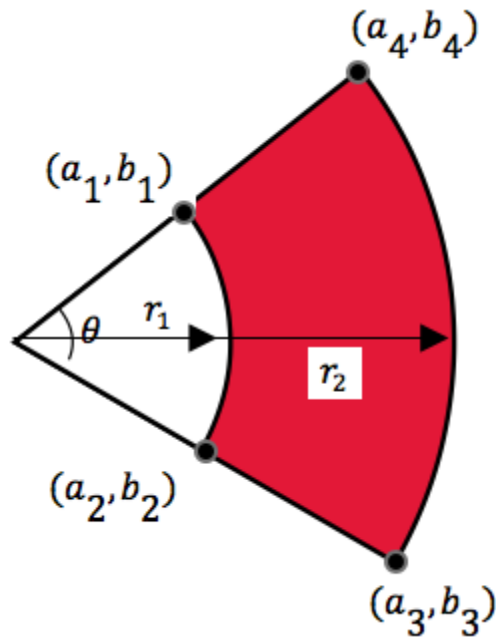
```
Rectangle = CCSCreateRectangle(list(0:0 10:10))  
Rectangle~>??  
;The segment property returns ccCurve segments of the rectangle  
Rectangle~>segments
```

When a polyCurve object is created using segmented curves, it is possible to apply a transformation to them. You can use this technique to create rhombus shapes by applying a transformation to a line segment. Rhombus curve segments can be created by generating a masterLine curve and then applying a proper transformation to create CC1, CC2, CC3, and CC4 as shown below.



Note: Counterclockwise rotation is described by positive sign and clockwise rotation is described by a negative sign.

To create pie polyCurve shapes, you first need to create a ccCurve object representing a circular arc. A pie polyCurve is composed of two circular arcs and two line-segments connecting these circular arcs.



It is simpler to represent the above arcs in cylindrical coordinates.

$$\begin{aligned} (r(t), \theta(t)) &= (r, t); \quad -\frac{\theta}{2} \leq t \leq \frac{\theta}{2} \\ (x(t), y(t)) &= (r(t) \sin(\theta(t)), r(t) \cos(\theta(t))) \end{aligned}$$

Considering that the parameterized curve should have Cartesian coordinates, you can define a utility function that converts the parametric form from cylindrical to Cartesian. The following function gets a parametric polar coordinates of a curve and returns its Cartesian coordinates.

```
procedure(CCSPolarToCartesian(ParameterizedPolar)
  let((Radius Theta ParamX ParamY)
  Radius = car(ParameterizedPolar)
  Theta = cadr(ParameterizedPolar)
  ParamX = lsprintf("%s*cos(%s)" Radius Theta)
  ParamY = lsprintf("%s*cos(%s)" Radius Theta)
  list(ParamX ParamY)
);let
);procedure
```

Virtuoso Photonics Solution Guide

CurvyCore Building Blocks

The following example defines the `CCSCreateArcSegment` function to create an arc. The function uses `CCSPolartoCartesian` described above to convert polar coordinates to Cartesian coordinates. The example defines two functions `CCSCreateArcSegment` and `CCSCreatePie`.

`CCSCreateArcSegment` creates a `ccCurve` object representing an arc. The operands are radius of the circle where `ArcLength` is 0. If the symmetric parameter is not `t`, the function will create an `ArcSegment` starting from 0.

```
procedure(CCSCreateArcSegment(Radius ArcLength @key(Symmetric t))
  let((ParameterizedArc ParameterizedPolar)
    ParameterizedPolar = list(lsprint("%n" Radius) lsprintf("%s" t))
    ; (R(t) Theta(t)) = (Radius t)
    ParameterizedArc.ParamX = car(CCSPolartoCartesian(ParameterizedPolar))
    ; (Radius t) --> (X(t), Y(t))
    ParameterizedArc.ParamY = cadr(CCSPolartoCartesian(ParameterizedPolar))
    if(Symmetric
      then
        ccCreateCurve(ParameterizedArc.ParamX ParameterizedArc.ParamY -ArcLength/
          2:ArcLength/2)e
      else
        ccCreateCurve(ParameterizedArc.ParamX ParameterizedArc.ParamY
          0:ArcLength)
    );if
  );let
);procedure
```

`CCSCreatePie` creates a `polyCurve` object representing a pie shape with positive area.

```
procedure(CCSCreatePie(Radius1 ArcLength1 Radius2 ArcLength2)
  let((ParameterizedPie ParameterizedArc1 LineSeg1 ParameterizedArc2 LineSeg2)
    ParameterizedArc1 = CCSCreateArcSegment(Radius1 -ArcLength1)
    ParameterizedArc2 = CCSCreateArcSegment(Radius2 ArcLength2)
    LineSeg1 = CCSCreateLineSegment(
      Radius1*cos(-ArcLength1/2):Radius1*sin(-ArcLength1/2)
      Radius2*cos(-ArcLength2/2):Radius2*sin(-ArcLength2/2))
    LineSeg2 = CCSCreateLineSegment(
      Radius2*cos(ArcLength1/2):Radius2*sin(ArcLength1/2)
      Radius1*cos(ArcLength2/2):Radius1*sin(ArcLength2/2))
    ccCreatePolyCurve(
      list(ParameterizedArc1 LineSeg1 ParameterizedArc2 LineSeg2)
      ?close t ?smoothness 'G0)
  );let;
);procedure
```

Related Information

[ccCreateCurve](#)

[ccCreatePolyCurve](#)

Paths and Facets

A path is a center curve that has parallel inner and outer curves. The center curve is defined as a ccCurve object. You can create a ccPath object using the [ccCreatePath](#) SKILL function. One of the uses of a ccPath object is to represent a photonics waveguide.

Consider that you want to create a 90° bend waveguide path with $0.5\mu\text{m}$ width and $15\mu\text{m}$ radius on a layer-purpose pair. The first step is to create the center ccCurve object of the path, as shown below.

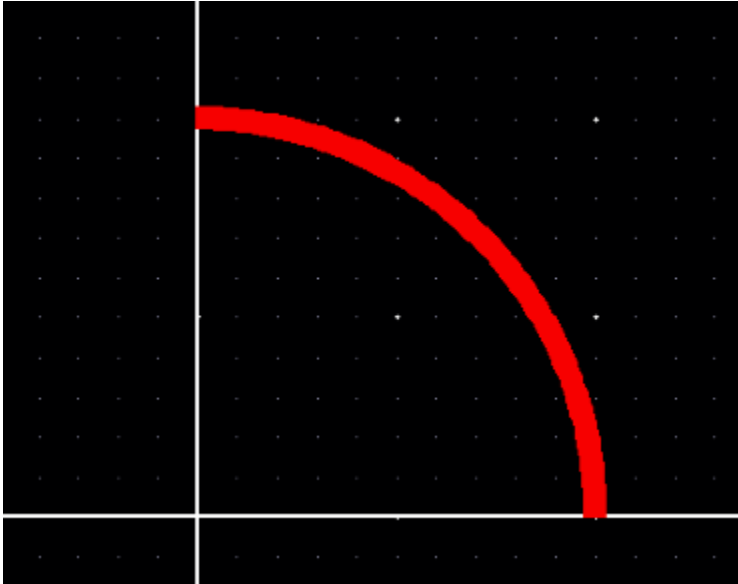
```
BendCenterCurve = CCSCreateArcSegment(15 asin(1))
```

The path needs to be associated with a layer in a technology file, therefore, you need to create a layout cell where the path can be created.

To generate a path object in the currently opened Layout Editor window, you can run the following command.

```
BendPath = ccCreatePath(geGetEditCellView() list("waveguide" "drawing")
    list(list(BendCenterCurve list(0:0 45) 0.5)))
;Generate the physical shape of BendPath
ccGenFigs(BendPath)
```

You should be able to see your 90° waveguide bend as shown below.



To generate physical shapes associated with a CurvyCore mathematical objects such as paths and surfaces, you need to use the [ccGenFigs](#) SKILL function.

Lets now create another ccPath object representing a straight waveguide with width 0.5μm.

```
StraightCenterCurve = CCSCreateLineSegment(0:0 10:0)
StraightPath = ccCreatePath(geGetEditCellView() list("waveguide" "drawing")
list(list(StraightCenterCurve list(0:0 0) 0.5)))
;Generate the physical shape of the path
ccGenFigs(StraightPath)
```

This example generates a straight path representing a straight photonic waveguide.



Attributes of ccPath Objects

A ccPath object consists of segments that are the center ccCurve objects with their associated inner and outer parallel curves. They also consists of two facets, `beginFacet` and `endFacet`.

Virtuoso Photonics Solution Guide

CurvyCore Building Blocks

```
StraightPath~>??
(cc:0x33874050 objType "ccPath" cellView db:0x20c7d39a
  lpp
  ("waveguide" "drawing") figs
  (db:0x20c78a9a) beginPt
  (0.0 0.0) endPt
  (10.0 0.0) beginFacet cc@0x68c1d0a6
  endFacet cc@0x47be58d4 segments
  ((cc@0xb16d5f44
    ((0.0 0.0) 0.0) "0.5"
  )
  ) smoothness
  G1
)
```

Facets are essential when ccPath objects need to be joined together at different levels of the hierarchy.

A facet is a mathematical object in the CurvyCore engine with following attributes:

- During physical shape generation, the center-point of a facet is always snapped to the manufacturing grid. This means that if a facet is rotated and the center point is not on the grid then the whole facet will be parallelly shifted to the nearest grid point.
- After the facet is shifted, the end points of the facet are shifted to the nearest points on the manufacturing grid. If there is an ambiguity to which point it should be shifted to (that is, if there are two or more nearest points on the manufacturing grid), then the end point is shifted to the farthest point from the origin of the facet (among those nearest points on the manufacturing grid).
- To avoid discontinuity, segments of boundary of the path or surface adjacent to the facet also need to be adjusted. Straight segments remain straight and are slightly transformed to connect to the new end point of the facet. Non-straight segments are adjusted by adding a nearly-zero fifth-degree polynomials to its x and y equations in such a way that a new segment is connected to the new end point of the facet, and the tangents and curvatures of the segment remain unchanged at the end points.

Surfaces and Facets

Mathematically, any simple-closed curve or polyCurve can be used to represent a surface.

When you go counter-clockwise on a closed surface, the enclosed area on the left-side of the boundary represents the surface having a positive surface area associated with that

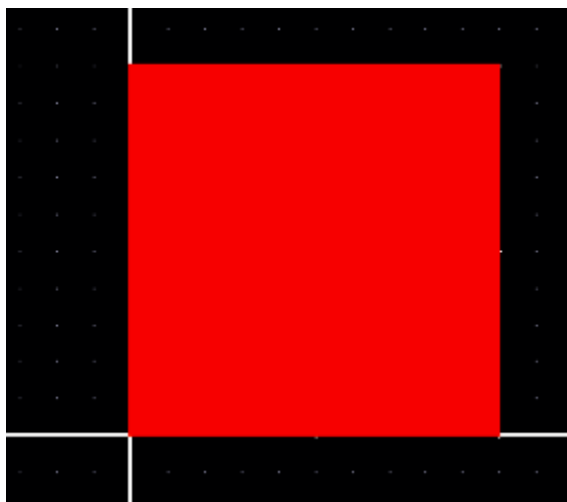
boundary. If you go clockwise, the associated surface is the same but the associated area is negative. A negative surface represents a hole that can be subtracted from another surface.

You can create CurvyCore surfaces using the `ccCreateSurface` SKILL function.

The following example creates a rectangular surface:

```
Rectangle = CCSCreateRectangle(list(0:0 10:10))
RectangleSurface = ccCreateSurface(
    geGetEditCellView()
    list("waveguide"
        "drawing")
    Rectangle)
ccGenFigs(RectangleSurface)
RectangleSurface~>??
(cc:0x2a0d3bf0 objType "ccSurface" cellView db:0x20c3d91a
 lpp
 ("waveguide" "drawing") figs
 (db:0x20c3c99a) area
 100.0 boundary cc@0x4e636b3f holes nil
)
```

This following figures shows the rectangular surface created using the above example.



The following example shows how to create a polygon with N number of sides or an *N-gon closed polyCurve* that can be used to create a hexagon surface.

```
procedure(CCSCreateNgon(Radius NumberofSides)
    let((Vertices(pi2*asin(1))))
    Vertices = tconc(nil Radius:0)
```

Virtuoso Photonics Solution Guide

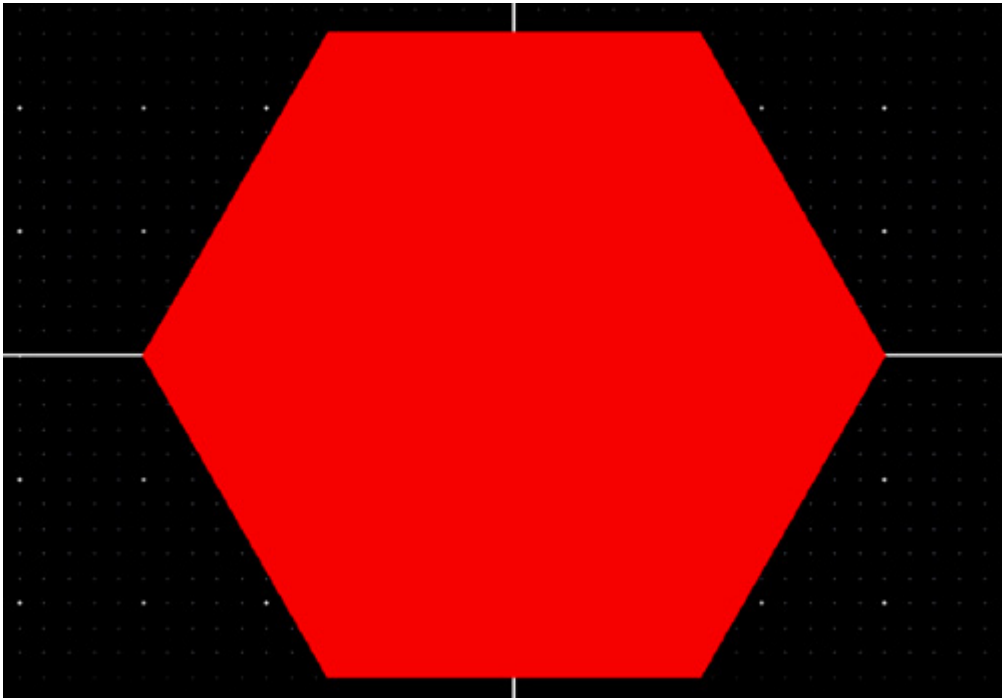
CurvyCore Building Blocks

```
for(i 1 NumberofSides-1
    tconc(
        Vertices
        Radius*cos(2*pi*i/NumberofSides):Radius*sin(2*pi*i/NumberofSides)
    );tonc
);for
Vertices = car(Vertices)
CCSCreatePolygon(Vertices)
);let
);procedure
```

Next, you can use the N-gon polyCurve to create a hexagon surface as shown below.

```
Hexagon = CCSCreateNgon(15 6)
HexagonSurface = ccCreateSurface(geGetEditCellView() list("waveguide" "drawing")
Hexagon)
ccGenFigs(HexagonSurface)
```

The figure shows the hexagon created using the above example.



The following procedure uses a closed pie polyCurve to create a pie surface.

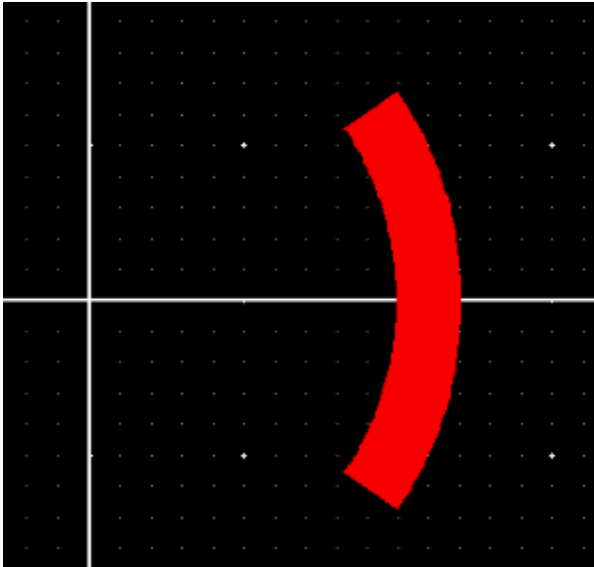
```
PiePolyCurve = CCSCreatePie(10 asin(1)*3/4 12 asin(1)*3/4)
PieSurface = ccCreateSurface(geGetEditCellView() list("waveguide" "drawing")
```


Virtuoso Photonics Solution Guide

CurvyCore Building Blocks

```
PiePolyCurve)  
ccGenFigs (PieSurface)
```

This figure shows pie surface created using a closed pie polyCurve.



Lets now see how to create a grating coupler shape as an array of grating pie shapes.

```
procedure (GratingCoupler (cv @key (designIntentLayer "waveguide")  
    (GratingNum 10)  
    (GratingSpace 3.0)  
    (GratingArcLength 2.0)  
    (Gratingwidth 1.0)  
    (GratingStartRadius 5.0)  
    (rootation 0.0)  
    (boundingBox t)  
)  
    let ((ccPolyCurves ccSurfaces BoundingBox BoundingBoxSurface  
        urx ury llx lly (pi asin(1)))  
  
    GratingArcLength = pi/GratingArcLength  
    declare (ccPolyCurves [GratingNum])  
    declare (ccSurfaces [GratingNum])  
  
    for (i 0 GratingNum-1  
        ccPolyCurves[i] = CCSCreatePie (  
            GratingStartRadius+GratingSpace*i  
            GratingArcLength  
            GratingStartRadius+GratingSpace*i+Gratingwidth  
            GratingArcLength)  
        ccPolyCurves[i] = ccTransformPolyCurve (ccPolyCurves[i]
```

Virtuoso Photonics Solution Guide

CurvyCore Building Blocks

```
list(0:0 rotation))

ccSurfaces[i] = ccCreateSurface(cv
                                list(designIntentLayer "drawing")
                                ccPolyCurves[i])
);for
when(boundingBox
    urx = GratingStartRadius+GratingSpace*(GratingNum-1) + Gratingwidth
    ury = (GratingStartRadius+GratingSpace*(GratingNum-1)) +
          Gratingwidth*sin(GratingArcLength/2)
    lly = -ury
    llx = GratingStartRadius*cos(GratingArcLength/2)
    BoundingBox = CSCCreateRectangle(list(llx:lly urx:ury))
    BoundingBoxSurface = ccCreateSurface(cv
                                          list("SiEtch0" "drawing")
                                          BoundingBox)
    ccMoveFig(BoundingBoxSurface ?transform list(0:0 rotation))
);when

list(ccSurfaces BoundingBoxSurface)
);let
);procedure
```

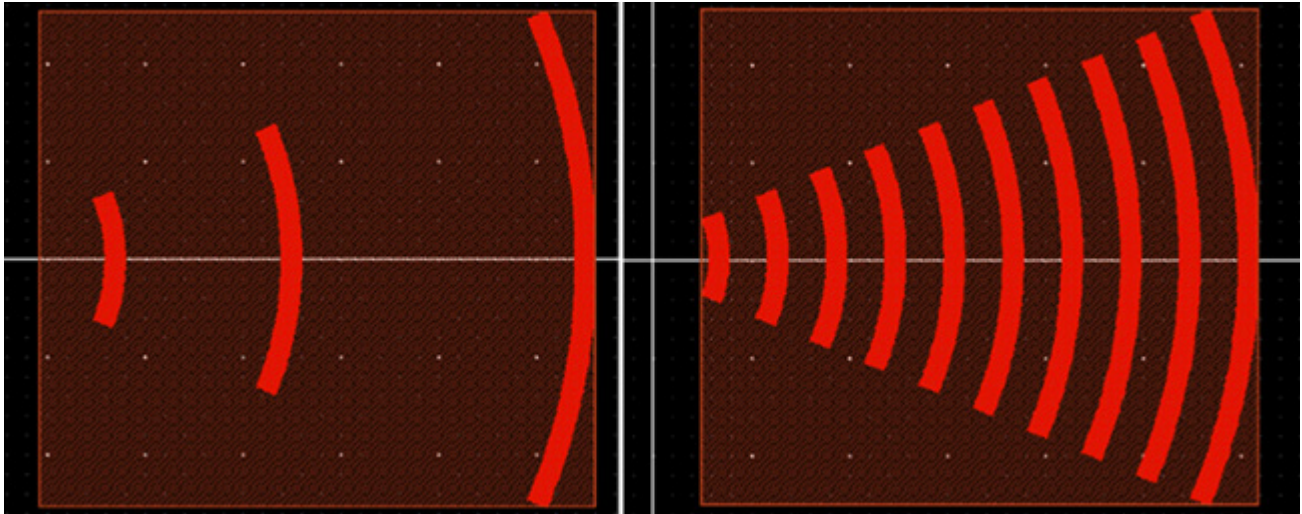
After the **GratingCoupler** function is loaded, you can check the creation of a grating coupler.

```
ccGrating = GratingCoupler(geGetEditCellView())
Gratings = car(ccGrating)
BoundingBox = cadr(ccGrating)
ccGenFigs(BoundingBox)
ccGenFigs(Gratings[1])
ccGenFigs(Gratings[4])
ccGenFigs(Gratings[9])
```

Virtuoso Photonics Solution Guide

CurvyCore Building Blocks

The following figures show the grating coupler shape with grating number 1, 4, 9, and the complete grating coupler.



A surface with a negative area represents a hole. This means that it is possible to subtract some holes from a surface when using `ccCreateSurface` SKILL function.

The holes that are going to be subtracted from the original surface are passed as a list parameter to the `ccCreateSurface` function as shown below.

```
procedure (GratingCouplerInverted (cv @key (designIntentLayer "waveguide")
    (GratingNum 10)
    (GratingSpace 3.0)
    (GratingArcLength 2.0)
    (Gratingwidth 1.0)
    (GratingStartRadius 5.0)
    (rotation 0.0)
    )
```

```
    let (
    (ccPolyCurves ccSurfaces BoundingBox BoundingBoxSurface
    urx ury llx lly (pi asin(1)))
```

```
    GratingArcLength = pi/GratingArcLength
```

```
    ;Create bounding box rectangle
```

```
    urx = GratingStartRadius+GratingSpace*(GratingNum-1)+Gratingwidth+1
```

```
    ury = (GratingStartRadius+GratingSpace*(GratingNum-
    1)+Gratingwidth)*sin(GratingArcLength/2)+1
```

```
    lly= -ury
```

```
    llx = GratingStartRadius*cos(GratingArcLength/2)-1
```

```
    BoundingBox = CCCreateRectangle(list(llx:lly urx:ury))
```

```
    for(i0 GratingNum-1
```

Virtuoso Photonics Solution Guide

CurvyCore Building Blocks

```
;ccPolyCurves are simple closed curves with negative area that represent
;the holes in the bounding box rectangle
ccPolyCurves=tconc(
    ccPolyCurves
    ccReverseCurve
    (CCSCreatePie(
        GratingStartRadius+GratingSpace*i
        GratingArcLength
        GratingStartRadius+GratingSpace*i+Gratingwidth
        GratingArcLength)
    )
);tconc
);for

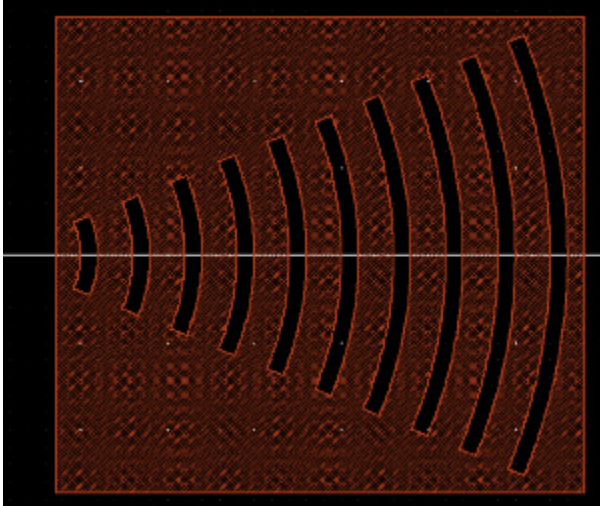
ccPolyCurves = car(ccPolyCurves)
ccSurfaces = ccCreateSurface(cv
    list("SiEtch0" "drawing")
    BoundingBox
    ccPolyCurves)

ccMoveFig(ccSurfaces ?transform list(0:0 rotation))

);let
);procedure
```

After defining and loading the `GratingCouplerInverted` function, you can create an instance where the gratings are subtracted from the bounding box.

```
ccGratingInverted = GratingCouplerInverted(geGetEditCellView())
ccGenFigs(ccGratingInverted)
```



Related Information

[ccCreatePath](#)

[ccGenFigs](#)

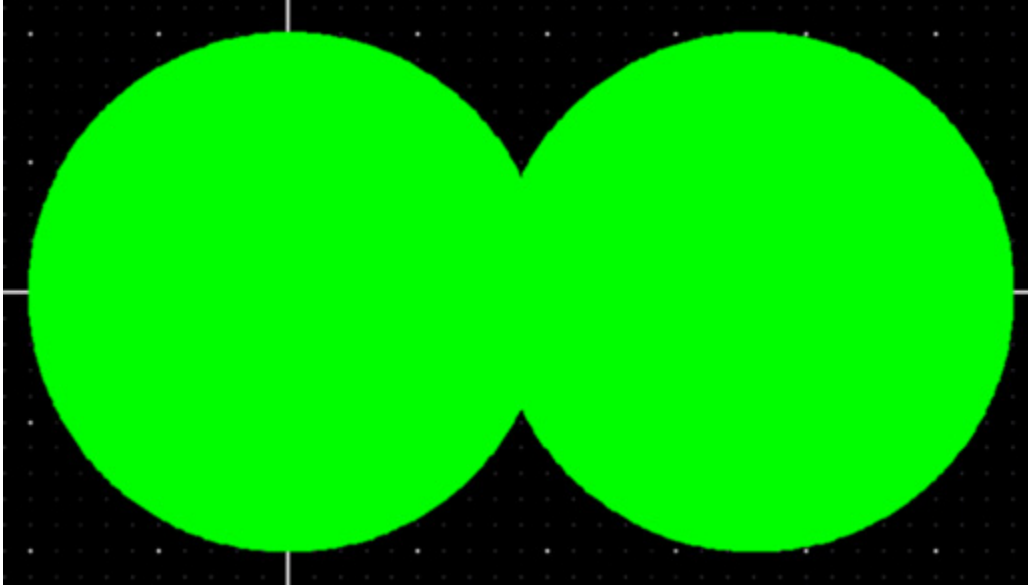
[ccCreateSurface](#)

Boolean Operations of Surfaces

Boolean operator `or` can be applied using the [ccLayerOr](#) SKILL function to create two circular surfaces and combine them to create a larger surface. The surfaces that this function acts on can be on different layers and the generated surface can also be in a different layer.

```
Circle = ccCreateCurve("10*cos(t)" "10*sin(t)" 0:4*asin(1))
Surf1 = ccCreateSurface(geGetEditCellView() list("waveguide" "drawing")
CirclePoly)
Surf2 = ccCreateSurface(geGetEditCellView() list("SiEtch0" "drawing")
ccTransformPolyCurve(CirclePoly list(18:0 0)))
SurfOr = ccLayerOr(Surf1 Surf2)
ccGenFigs(car(SurfOr))
```

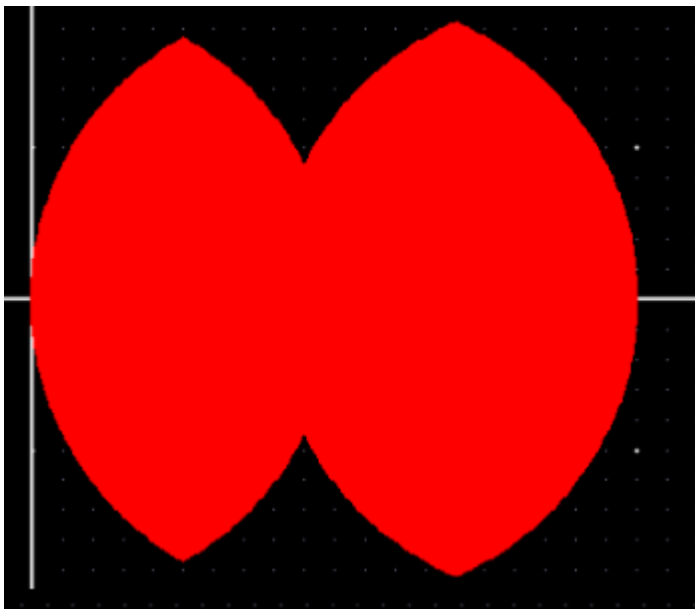
The following figure shows how two circular surfaces are combined to create a larger surface using the `ccLayerOr` function.



Boolean operator `and` can be applied using the `ccLayerAnd` SKILL function to extract the overlap area of the above shape with a transformed circle surface.

```
Surf3 = ccCreateSurface (geGetEditCellView() list("waveguide" "drawing")
ccTransformPolyCurve (CirclePoly list(10:0 0)))
SurfAnd = ccLayerAnd(SurfOr Surf3 ?lpp list("wavgeuide" "drawing"))
ccGenFigs (car (SurfAnd))
```

In this figure, you can see that the overlap area of the shape has been extracted from the shape.



Related Information

[ccLayerOr](#)

[ccLayerAnd](#)

Smoothing of Surfaces

You can use the [ccSmoothenFig](#) SKILL function to smoothen a surface. Assume that you have a hexagonal surface, and that at the sharp corners of the hexagon, the bend radius is zero. Now assume that you want to smoothen the shape and specify the minimum radius of curvature for the shape to be 4.

In the example below you get similar results.

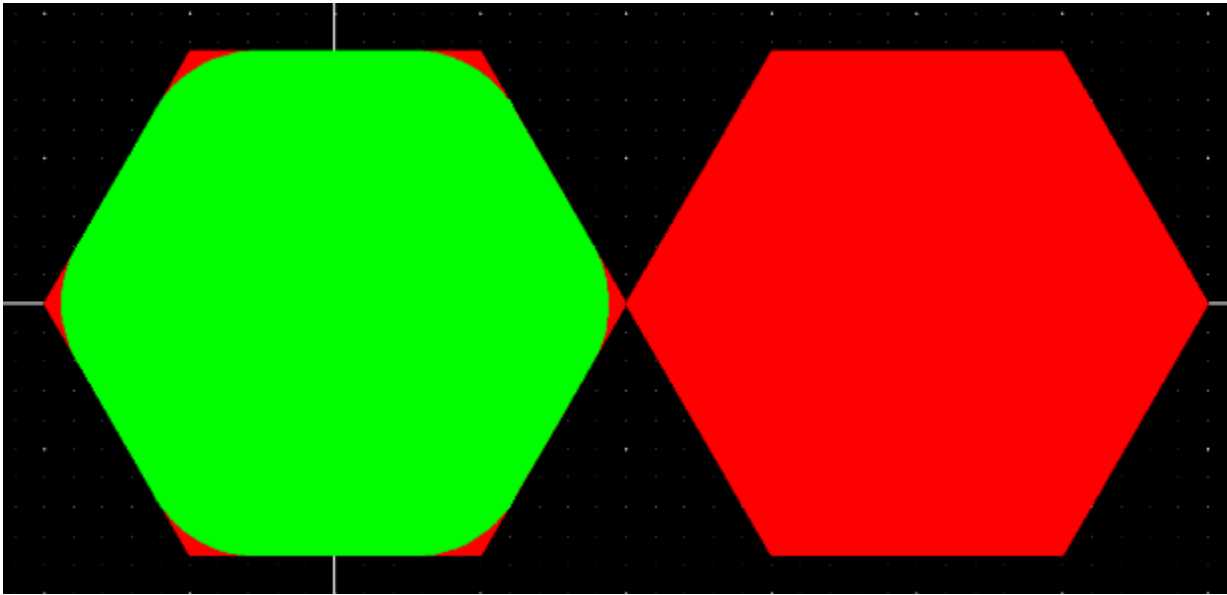
```
Hex = CCSCreateNgon(10 6)
HexSurf = ccCreateSurface(geGetEditCellView() list("waveguide" "drawing") Hex)
Hex2Surf = ccCreateSurface(geGetEditCellView() list("waveguide" "drawing")
ccTransformPolyCurve(Hex list(20:0 0)))
Hex3 = ccCreateSurface(geGetEditCellView() list("SiNWaveguide" "drawing") Hex)
HexSmooth = ccSmoothenFig(Hex3 -4)
ccGenFigs(HexSurf)
ccGenFigs(Hex2Surf)
```

Virtuoso Photonics Solution Guide

CurvyCore Building Blocks

`ccGenFigs (HexSmooth)`

In this figure, there are three hexagonal surfaces, the red surface on the right is the original hexagonal surface and the two overlapped surfaces are the same original hexagonal surface and its smoothed version.



Related Information

[ccSmoothenFig](#)

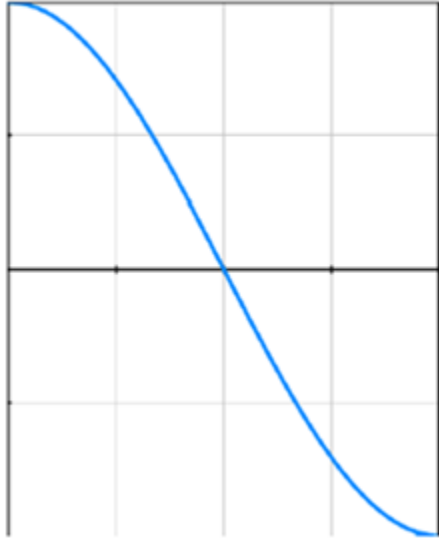
CurvyCore Pcells

There is no significant difference between a typical Pcell and a CurvyCore Pcell other than the fact that in a CurvyCore Pcell, you can use the CurvyCore and photonic functions to generate and manipulate your shapes. This means that all the concepts and best practices of creating a Pcell also work in this situation.

Virtuoso Photonics Solution Guide

CurvyCore Building Blocks

To understand how to work with CurvyCore Pcells, let's consider an S-bend connector Pcell. The parameterized equation and the shape of this connector is shown below.



$$x(t) = t$$
$$y(t) = \frac{Height}{2} * \cos\left(\frac{t * \pi}{Length}\right)$$
$$0 \leq t \leq 4 ; Height = 4 ; Length = 4$$

Let's see how to create a generator for the S-bend shape and then use that inside the Pcell code to create the ccPath object.

The following procedure creates a generator for cosine S-bend curve:

```
procedure(CCSCreateSBend(height length)
  let((parameterizedArc)
    parameterizedArc.ParamX = "t"
    parameterizedArc.ParamY = lsprint("%n/2*cos(t*pi/%n)" height length)
    ccCreateCurve(parameterizedArc.ParamX parameterizedArc.ParamY 0:length)
  );let
);procedure
```

In the following Pcell code, width parameter specifies the width of the waveguide path and rotation parameter specifies the rotation applied to the final waveguide path. Next, S-bend generator is used to create the S-bend curve and build the waveguide path.

```
pcDefinePCell(
  list(ddGetObj("gopdk") "CustomSBend" "layout")
    ;Pcell parameters
    (
      (designIntentLayer "string" "waveguide")
      (rotation "string" "0.0")
      (height "string" "10.0u")
      (length "string" "40.0u")
      (width "string" "0.5u")
    );end of parameters
```

Virtuoso Photonics Solution Guide

CurvyCore Building Blocks

```
;Pcell body code
let((ccCurve ccPath ccWaveguide)
    height = readstring(pcCellView~>parameters~>height)*1e6;
    length = readstring(pcCellView~>parameters~>length)*1e6;
    width = readstring(pcCellView~>parameters~>width)*1e6;
    rotation = readstring(pcCellView~>parameters~>rotation);
);let
);pcDefinePCell
```

The following code adds three CDF parameters and sets how interpreted labels should be evaluated. Populate the CDF data after loading the file.

Note: Pcell parameter names and their associated CDF property names should be the same.

```
let((cellId cdfId)
    cellId = ddGetObj("gopdk" "CustomSBend")
    when(cdfId = cdfGetBaseCellCDF(cellId)
        cdfDeleteCDF(cdfId)
    );when

cdfId = cdfCreateBaseCellCDF(cellId)
cdfCreateParam(cdfId
    ?name      "width"
    ?prompt    "Waveguide Width(M) "
    ?defValue  "0.5u"
    ?type      "string"
    ?storeDefault"t")

cdfCreateParam(cdfId
    ?name      "length"
    ?prompt    "Bend Length(M) "
    ?defValue  "40.0u"
    ?type      "string"
    ?storeDefault"t")

cdfCreateParam(cdfId
    ?name      "height"
    ?prompt    "Bend Height(M) "
    ?defValue  "10.0u"
    ?type      "string"
    ?storeDefault"t")

;These are related to Cell/Instance Name(cdsName)
cdfId->instDisplayMode = "instName"
cdfId->instNameType = "schematic"
;These are related to terminals(cdsTerm)
cdfId->termDisplayMode="natName"
cdfId->netNameType="schamatic"
```

Virtuoso Photonics Solution Guide

CurvyCore Building Blocks

```
;These are related to Parameters(cdsParam)
cdfId->paramDisplayMode="parameter"
cdfId->paramLabelSet="width length height"

cdfSaveCDF(cdfId)
);let
```

When waveguideTemplates are defined in the technology database, it is possible to use them to generate a waveguide from a ccPath object.

```
("Deep SOI Waveguide" ("waveguide" "drawing")
  'minWidth      0.5
  'minBendRadius 10.0
  'modeProperties(
    'function "gopdkModePropWG"
    'namespace(("TE 0")("TM" 1))
  )
  'derivedShapeSpecs(
    ( "SiEtch0" 'type "enclosure" 'enclosure 0.05)
    ( "SiEtch2" 'type "enclosure" 'enclosure 2.0)
  )
)
```

Next, call phoGenWaveguide to generate the required derived shapes.

```
cdfCreateParam(cdfId
  ?name      "generateWaveguide"
  ?prompt    "Generate Waveguide Derived Shapes"
  ?defValue  "t"
  ?parseAsCel "no"
  ?type      "boolean"
  ?storeDefault "t"
  :storeDefault "yes"
  ?display    "t")

pcDefinePCell(
  list(ddGetObj("gopdk") "CustomSBend" "layout")
    ;Pcell parameters
    (
      (designIntentLayer "string" "waveguide")
      (rotation "string" "0.0")
      (height "string" "10.0u")
      (length "string" "40.0u")
      (width "string" "0.5u")
      (generateWaveguide "boolean" t)
    );end of parameters
  ;Pcell body code
  let((ccCurve ccPath ccWaveguide)
```

Virtuoso Photonics Solution Guide

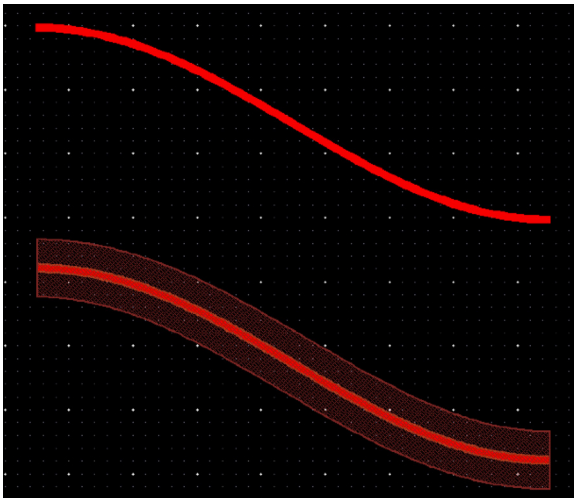
CurvyCore Building Blocks

```
height = readstring(pcCellView~>parameters~>height)*1e6;
length = readstring(pcCellView~>parameters~>length)*1e6;
width = readstring(pcCellView~>parameters~>width)*1e6;
rotation = readstring(pcCellView~>parameters~>rotation);

ccCurve = CCSCreateSBend(height length)
ccpath = ccCreatePath(
    pcCellView
    list(designIntentLayer "drawing")
    list(list(ccCurve list(0:0 rotation)width)))
ccGenFigs(ccpath)

when(generateWaveguide
    ccWaveguide=phoGenWaveguide(ccPath)
    foreach(CCObj ccWaveguide
        ccGenFigs(CCObj)
    );foreach
);when
);let
);pcDefinePCell
```

The following figure shows the S-bend generated on the design intent layer (top shape) and the same S-bend when its associated derived waveguide layers are generated (bottom shape).



Related Information

[Photonics Functions](#)

[Waveguide Connectors](#)

[Virtuoso Parameterized Cell Reference](#)

Waveguide Connectors

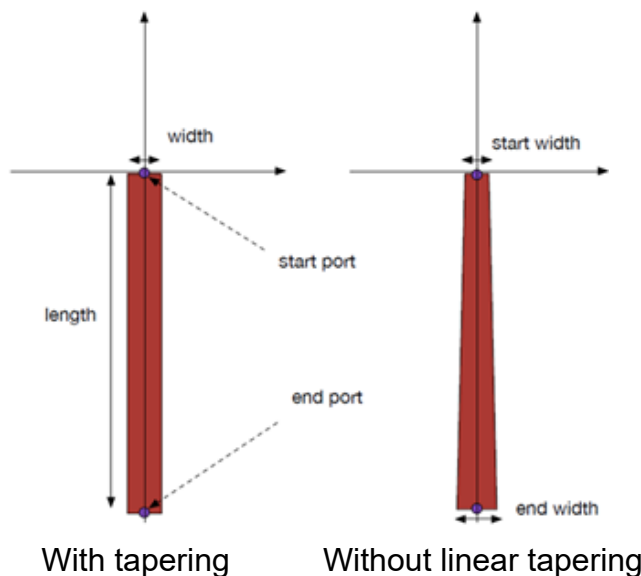
CurvyCore building blocks include a set of waveguide connectors such as straight, bend, curve, sine, and waypoint connectors. These connectors are two-port optical waveguides that are used as connector between other optical devices.

CurvyCore building blocks have a set of common features and parameters. Two of these parameters are offset and rotation.

Straight Connectors

The general shape of a straight waveguide connector is shown in the following figure.

General shape of a straight waveguide connector



A straight waveguide connector supports two type of straight connections, with tapering and without tapering. It further supports four types of tapering, linear, exponential, logarithmic, and parabolic. If tapering is enabled, you need to set the start and end width and also type of taper

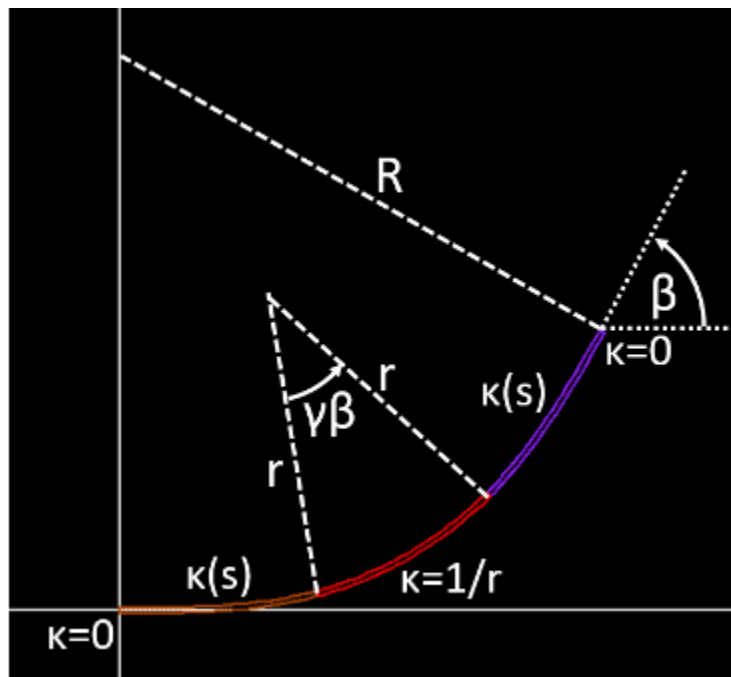
when you call the generator, otherwise, width specifies the default width of the straight waveguide. It also supports parameters such as start port and end port names.

Related Information

Photonics Functions

Bend Connectors

A bend connector connects two points on the same circle with angles equal to the angles of the tangents to that circle. The bend connector is described by the bend radius R of the circle and the bend angle of the arc of that circle. In its simplest form, a bend connector consists of a single circular arc segment. If the bend connector connects two straight waveguide segments with zero curvature $\kappa = 0$, the curvature at the ports changes discontinuously, causing additional connection loss. This curvature discontinuity can be avoided if additional curve segments are inserted between the straight line segments and the circular arc segments so that the curvature becomes a continuous function $\kappa(s)$ of the arc length as shown in the following figure.



Structure of a bend connector

Related Information

[ccBendConnector](#)

Curve Connectors

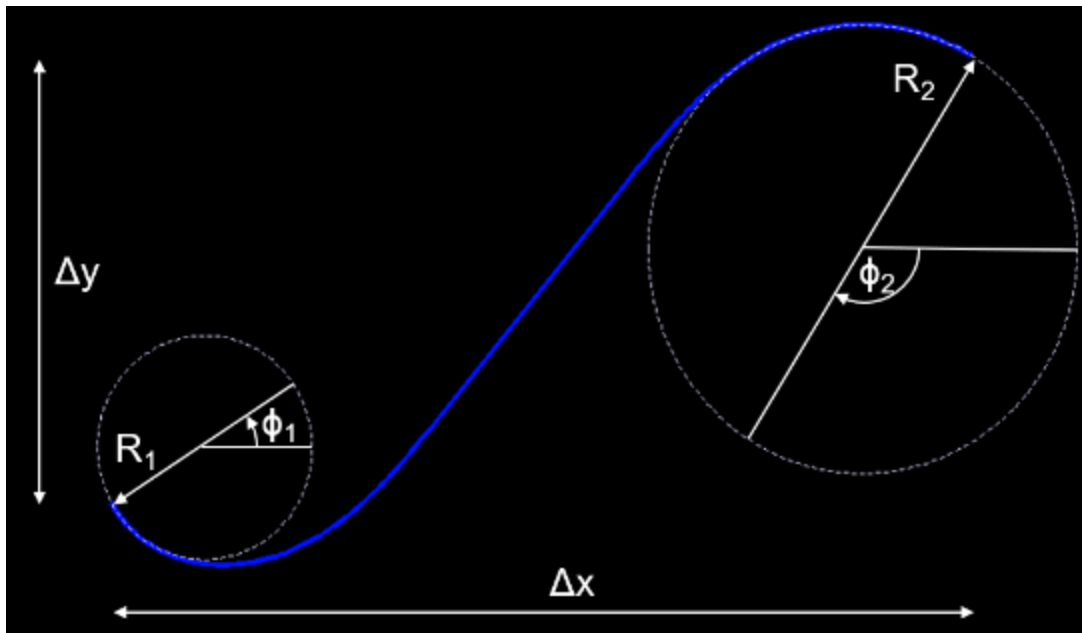
A curve connector connects one port with the following specifications:

- coordinates: x_1, y_1
- facet angle: ϕ_1
- curvature radius: R_1

To another port with the following specifications:

- coordinates: x_2, y_2
- facet angle: ϕ_2
- curvature radius: R_2

Curve connectors connect two points with a set of constraints on the facet angles and curvature radius as shown in the following figure:



If the coordinate system is rotated and translated such that one port is at the origin and another port is on the x-axis, the curve connector is described in terms of the distance

between the ports, the two facet angles with respect to the x-axis, and the two curvature radii. These five boundary conditions together with the minimum curvature radius R_{min} represent six constraints for the calculation of the curve connector. With only six constraints, the calculation of curve connectors is an under-constrained issue. One additional constraint can be imposed by requiring that:

$$d^2\kappa(s)/d^2s = 0$$

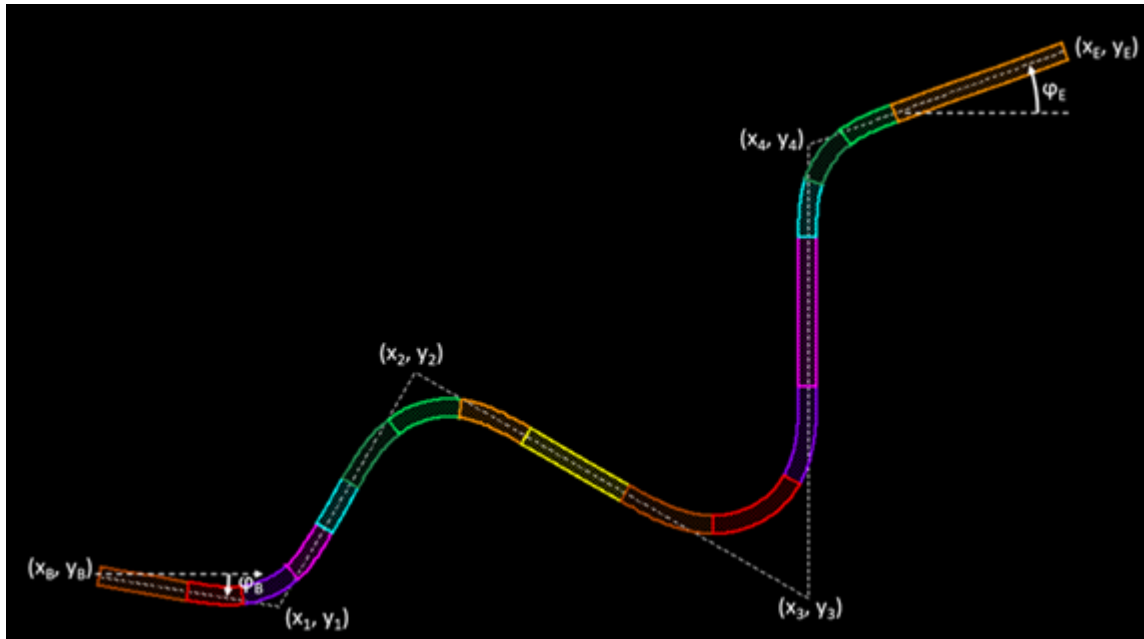
This limits the solution space to only three classes of curves, namely straight lines, circular arcs and clothoids. Curve connectors are thus a special case of clothoid splines.

Related Information

[ccCurveConnector](#)

Waypoint Connectors

Waypoint connector connects the begin port with coordinates (x_b, y_b) to the end port with coordinates (x_E, y_E) by a sequence of straight line segments and bend connectors. The straight line segments are defined by a polygonal line through the series of waypoints with coordinates (x_i, y_i) as shown in the following figure:



Structure of a waypoint connector

The facet angles at the ports are determined by the angles φ_b and φ_E of the lines connecting the begin point with the first waypoint and the end point with the last waypoint, respectively.

Similarly to a single bend connector, each bend connector in a waypoint connector can also consist of a circular arc connected to the straight lines at its ends by a pair of apex clothoids. This is also shown in the above figure, with the straight line segments, the circular arc segments, and the clothoid segments displayed in different colors. The waypoint connector can be calculated using the [ccWaypointConnector](#) SKILL function.

Related Information

[ccWaypointConnector](#)

Virtuoso Photonics Solution Guide

CurvyCore Building Blocks

Photonics Environment Variables

This appendix provides information on the environment variable names, descriptions, and graphical user interface equivalents for the Virtuoso[®] Photonics Solution.

Note: Only the environment variables documented in this chapter are supported for public use. All other Photonics Solution environment variables, regardless of their name or prefix, and undocumented aspects of the environment variables that are described below are private and are subject to change at any time.

Related Topics

- [Setting Environment Variables](#)
- [List of Photonics Solution Environment Variables](#)

Setting Environment Variables

Environment variables control the values of the Virtuoso Photonics Solution options.

For information on setting the environment variables, see [Setting Environment Variables](#).

For a list of all the supported environment variables and their values, see [List of Photonics Solution Environment Variables](#).

List of Photonics Solution Environment Variables

cweAlignElements

cweMatchWidths

opticalSigTypePropagation

opticalNetColor

opticalNetColoring

opticalNetLineStyle

phoAbutAutoAdjust

phoAbutClass

phoAbutFunction

phoAbutNonPcells

phoComposeMaster

phoComposeMasterPromptOff

phoPinInputAngle

phoPinLayer

phoPinRadius

phoPinWidth

photonicDisplay

photonicPinWidth

photonicPinAngle

photonicPinFacetInPacket

photonicPinFacetOutPacket

photonicPinLabelPacket

Virtuoso Photonics Solution Guide

Photonics Environment Variables

photonicPinRadius

photonicPinRadiusLinePacket

photonicPinWidthLinePacket

srcOpticalElectricalConnection

srcOpticalMultiToSingle

srcOpticalSingleToMulti

srcOpticalTooManyConnections

cweAlignElements

```
photronics cweAlignElements boolean { t | nil }
```

Description

Rotates the subsequent components inside a composite waveguide to maintain their alignment when the selected component is edited to change its rotation, total angle, or radius. The changes in rotation are only propagated forward to the subsequent components of the edited component, not propagated back to the components that exist earlier in the composite. For example, if a composite is made up of components in the sequence: Opt1 -> Opt2 -> Opt3, editing the rotation of Opt2 will update the rotation of Opt3 but it will not update the rotation of Opt1.

When matching the alignment of the waveguide components, the `rotateInstParam` environment variable is used to specify the name of the parameter to use to control the rotation of the waveguide components. The parameter name is first searched in the CDF parameter list. If a corresponding CDF parameter that can control the rotation of the waveguide components is not found, a corresponding property associated with the waveguide components is used.

The default is `t`.

When set to `nil`, the rotation of the subsequent components is not automatically adjusted to match the edited component and manual editing of individual components is required to set all the components to the correct rotation.

GUI Equivalent

Command	<i>Composite Waveguide Editor – Editor Options</i>
Field	<i>Align elements</i>

Examples

```
envGetVal("photronics" "cweAlignElements")
envSetVal("photronics" "cweAlignElements" 'boolean nil)
```

Related Topics

[ciGetParamMapping](#)

Virtuoso Photonics Solution Guide

Photonics Environment Variables

[rotateInstParam](#)

[dbGetPropByName](#)

List of Photonics Solution Environment Variables

Setting Environment Variables

cweMatchWidths

```
photronics cweMatchWidths boolean { t | nil }
```

Description

Updates the width of components inside a composite waveguide to ensure the width of each subsequent component matches that of the edited waveguide component. Any changes in width to an edited component are not propagated back to the components that exist earlier in the composite. For example, if a composite is made up of components in the sequence: Opt1 -> Opt2 -> Opt3, editing the width of Opt2 will update the width of Opt3 but it will not update the width of Opt1.

The default is `t`.

When set to `nil`, the width of the subsequent components is not automatically adjusted to match the edited component and manual editing of individual components is required to set all the components to the correct width.

GUI Equivalent

Command	<i>Composite Waveguide Editor – Editor Options</i>
Field	<i>Match Widths</i>

Additional Information

When matching the component widths for a composite waveguide, different width parameters are considered for tapered and non-tapered devices.

■ Non-tapered devices

When matching width for a non-tapered device in a composite waveguide, the parameter on the device that matches one of the following in the given order gets used:

- a. Parameter defined in the `ciGetParamMapping("waveguideWidth")` SKILL function.
- b. Parameter called `"width"`, `"wgWidth"` or `"wg_width"`
- c. Regular expression for a parameter with `"width"` in its name

■ Tapered devices

Virtuoso Photonics Solution Guide

Photonics Environment Variables

When matching width for a tapered component in a composite waveguide, the matched width is based either on the `startWidth` or the `endWidth` of the taper. The parameter on the device that matches one of the following in the given order gets used:

- a. Parameter defined in the `ciGetParamMapping("startFacetWidth")` or `ciGetParamMapping("endFacetWidth")` **SKILL** function.
- b. Regular expression for a parameter that matches `("width.*in|start|begin")` or `("width.*out|end")`

Examples

```
envGetVal("photronics" "cweMatchWidths")
envSetVal("photronics" "cweMatchWidths" 'boolean nil)
```

Related Topics

[ciGetParamMapping](#)

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

opticalSigTypePropagation

```
schematic opticalSigTypePropagation boolean { t | nil }
```

Description

Propagates the optical signal type.

The default is `t`.

GUI Equivalent

Command

Options – Check

Form Field

Propagate the Optical Signal Type Values ([Schematic Check Options](#) form)

Examples

```
envGetVal("schematic" "opticalSigTypePropagation")
envSetVal("schematic" "opticalSigTypePropagation" 'boolean t)
envSetVal("schematic" "opticalSigTypePropagation" 'boolean nil)
```

Related Topics

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

opticalNetColor

```
schematic opticalNetColor string "cream"
```

Description

Specifies the color of the optical nets.

The line style for the optical nets can also be specified using [opticalNetLineStyle](#).

GUI Equivalent

Command	<i>Options – Check</i>
Form Field	<i>Set Optical Wire Style (Schematic Check Options form)</i>

Examples

```
envGetVal("schematic" "opticalNetColor")  
envSetVal("schematic" "opticalNetColor" 'string "blue")
```

Related Topics

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

opticalNetColoring

```
schematic opticalNetColoring boolean { t | nil }
```

Description

Specifies whether the optical nets need to have a specified color.

The default is `t`.

GUI Equivalent

None

Examples

```
envGetVal("schematic" "opticalNetColoring")  
envSetVal("schematic" "opticalNetColoring" 'boolean t)  
envSetVal("schematic" "opticalNetColoring" 'boolean nil)
```

Related Topics

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

opticalNetLineStyle

```
schematic opticalNetLineStyle string "thickLine"
```

Description

Specifies the line style for the optical nets.

The color for the optical nets can also be specified using [opticalNetColor](#).

GUI Equivalent

Command	<i>Options – Check</i>
Form Field	<i>Set Optical Wire Style (Schematic Check Options form)</i>

Examples

```
envGetVal("schematic" "opticalNetLineStyle")  
envSetVal("schematic" "opticalNetLineStyle" 'string "thinline")
```

Related Topics

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

phoAbutAutoAdjust

```
photronics phoAbutAutoAdjust boolean { t | nil }
```

Description

Automatically rotates the moving waveguide to match the rotation of the abutting interface. When the moving object is a top-level pin, the pin `width`, `radius`, and `angle` are adjusted to allow abutment without requiring manual adjustment.

The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("photronics" "phoAbutAutoAdjust")  
envSetVal("photronics" "phoAbutAutoAdjust" 'boolean t)
```

Related Topics

[Abutting Photonic Elements](#)

phoAbutClass

```
layoutXL phoAbutClass string "valid_abutment_class"
```

Description

Specifies that two optical pins can abut only when they belong to the same abutment class. Examples of abutment classes that can be used for abutting pins include the pin layer, pin edge, and so on.

The default is " ", which means that the pin layer is used to get the `abutClass` property because the property is neither set on the Pcell, nor defined by this environment variable.

Note:

- When the environment variable is set to t, the pin layer refers to the layer, irrespective of the layer purpose.
- If the Pcell or non-Pcell instances do not have the `abutClass` abutment property set on the abutting pinFigs, the value set for the `phoAbutClass` environment variable is used for abutment.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "phoAbutClass")  
envSetVal("layoutXL" "phoAbutClass" 'string "pin_layer")
```

Related Topics

[Abutment in Virtuoso Photonics Solution](#)

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

phoAbutFunction

```
layoutXL phoAbutFunction string "valid_function_name"
```

Description

Specifies a user-defined callback to process abutment and unabutment.

The default is "phoWGAbut", which means that this function is used to abut Pcells, non-Pcells, or both in photonic designs.

Note: If the Pcell or non-Pcell instances do not have the abutment property set on the abutting pinFigs, the value set for the `phoAbutFunction` environment variable is used for abutment.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "phoAbutFunction")  
envSetVal("layoutXL" "phoAbutFunction" 'string "phoWGAbut")
```

Related Topics

[Abutment in Virtuoso Photonics Solution](#)

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

phoAbutNonPcells

```
layoutXL phoAbutNonPcells boolean { t | nil }
```

Description

Controls whether the abutment of photonic non-Pcell instances is supported.

The default is "t", which means:

- Top-level pins can abut with instance pins.
- Photonic non-Pcell instances can abut with both non-Pcell and Pcell instances.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "phoAbutNonPcells")  
envSetVal("layoutXL" "phoAbutNonPcells" 'boolean t)  
envSetVal("layoutXL" "phoAbutNonPcells" 'boolean nil)
```

Related Topics

[Abutment in Virtuoso Photonics Solution](#)

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

phoComposeMaster

```
layoutXL phoComposeMaster string "master_cellview_name"
```

Description

Specifies the library cellview of the cell master to be used for creating the composite waveguide.

The default is " "

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "phoComposeMaster")  
envSetVal("layoutXL" "phoComposeMaster" 'string "pholib waveMaster layout")
```

Related Topics

[phoComposeMasterPromptOff](#)

[Abutment in Virtuoso Photonics Solution](#)

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

phoComposeMasterPromptOff

```
layoutXL phoComposeMasterPromptOff boolean { t | nil }
```

Description

Uses the bound schematic instances, if available, to determine the master library cellview to use for creating the composite waveguide. If the bound schematic instances are not available, uses a previous selection to determine the master cellview to use.

The default is "nil", which means a form is raised that enables you to specify the master library cellview to be used.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "phoComposeMasterPromptOff")  
envSetVal("layoutXL" "phoComposeMasterPromptOff" 'boolean t)  
envSetVal("layoutXL" "phoComposeMasterPromptOff" 'boolean nil)
```

Related Topics

[phoComposeMaster](#)

[Abutment in Virtuoso Photonics Solution](#)

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

phoPinInputAngle

layoutXL phoPinInputAngle float *floating_point_number*

Description

Specifies the default angle to use for generating the optical input pins in the layout.

The default is 90.0.

Arguments

floating_point_number,

A floating point number that specifies the default angle to use for creating the optical input pins in the layout.

GUI Equivalent

Command: *Connectivity – Generate – All From Source – I/O Pins* (tab)

Field: *Default Values for Optical Pins – Input Angle* (field)

Examples

```
envGetVal("layoutXL" "phoPinInputAngle")  
envSetVal("layoutXL" "phoPinInputAngle" 'float 45.0)
```

Related Topics

[Generating Optical Pins](#)

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

phoPinLayer

```
layoutXL phoPinLayer string "valid_photonic_layer_name"
```

Description

Specifies the default photonic layer to use for generating optical pins in the layout.

The default is "".

If no default layer is specified in the environment variable, the first photonic layer in the list of `validLayers` is used for optical pin generation.

GUI Equivalent

Command: *Connectivity – Generate – All From Source – I/O Pins* (tab)

Field: *Default Values for Optical Pins – Layer* (field)

Examples

```
envGetVal("layoutXL" "phoPinLayer")  
envSetVal("layoutXL" "phoPinLayer" 'string "waveguide drawing")
```

Related Topics

[Generating Optical Pins](#)

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

phoPinRadius

layoutXL phoPinRadius float *floating_point_number*

Description

Specifies the default radius to use for generating optical pins in the layout.

The default is 0.0.

Arguments

floating_point_number,

A floating point number that specifies the default radius for creating the circular optical pins in the layout.

GUI Equivalent

Command: *Connectivity – Generate – All From Source – I/O Pins* (tab)

Field: *Default Values for Optical Pins – Radius* (field)

Examples

```
envGetVal("layoutXL" "phoPinRadius")  
envSetVal("layoutXL" "phoPinRadius" 'float 0.2)
```

Related Topics

[Generating Optical Pins](#)

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

phoPinWidth

layoutXL phoPinWidth float *floating_point_number*

Description

Specifies the default width to use for generating optical pins in the layout.

The default is 0.0.

Arguments

floating_point_number,

A floating point number that specifies the default width for creating the optical pins in the layout.

GUI Equivalent

Command: *Connectivity – Generate – All From Source – I/O Pins* (tab)

Field: *Default Values for Optical Pins – Width* (field)

Examples

```
envGetVal("layoutXL" "phoPinWidth")  
envSetVal("layoutXL" "phoPinWidth" 'float 0.2)
```

Related Topics

[Generating Optical Pins](#)

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

photonicDisplay

```
graphic photonicDisplay boolean { t | nil }
```

Description

Controls whether to display photonic pin attributes in layout (t) or not (nil).

The default is nil.

GUI Equivalent

None

Examples

```
envGetVal("graphic" "photonicDisplay")  
envSetVal("graphic" "photonicDisplay" 'boolean t)  
envSetVal("graphic" "photonicDisplay" 'boolean nil)
```

Related Topics

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

photonicPinWidth

layout photonicPinWidth float *float_number*

Description

Specifies the width of the facet of the waveguide that connects to a pin. The default value is 0.1.

GUI Equivalent

Command	<i>Create – Pin – Manual</i>
Field	<i>Photonic Width</i> (Create Pin Form)

Examples

```
envGetVal("layout" "photonicPinWidth")  
envSetVal("layout" "photonicPinWidth" 'float 0.5)
```

Related Topics

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

photonicPinAngle

```
layout photonicPinAngle float float_number
```

Description

Specifies the angle of the facet of the waveguide at the intersection of the pin.

The default is 0.0.

GUI Equivalent

Command	<i>Create – Pin – Manual</i>
Field	<i>Photonic Angle (Create Pin Form)</i>

Examples

```
envGetVal("layout" "photonicPinAngle")  
envSetVal("layout" "photonicPinAngle" 'float 15)
```

Related Topics

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

photonicPinFacetInPacket

`graphic photonicPinFacetInPacket string packetName`

Description

Specifies the name of the packet for the pin on which a waveguide facet is drawn inside.

GUI Equivalent

None

Examples

```
envGetVal("graphic" "photonicPinFacetInPacket")  
envSetVal("graphic" "photonicPinFacetInPacket" 'string "phoPinFacetIn")
```

Related Topics

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

photonicPinFacetOutPacket

`graphic photonicPinFacetOutPacket string packetName`

Description

Specifies the name of the packet for the pin on which a waveguide facet is drawn outside.

GUI Equivalent

None

Examples

```
envGetVal("graphic" "photonicPinFacetOutPacket")
envSetVal("graphic" "photonicPinFacetOutPacket" 'string "phoPinFacetOut")
```

Related Topics

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

photonicPinLabelPacket

`graphic photonicPinLabelPacket string packetName`

Description

Specifies the name of the packet in which the photonic pin label is drawn.

GUI Equivalent

None

Examples

```
envGetVal("graphic" "photonicPinLabelPacket")  
envSetVal("graphic" "photonicPinLabelPacket" 'string "phoPinLabel")
```

Related Topics

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

photonicPinRadius

```
layout photonicPinRadius float float_number
```

Description

Specifies the radius of the curve of the waveguide center at the point it meets the facet.

The default value is 0.0.

GUI Equivalent

Command	<i>Create – Pin – Manual</i>
Field	<i>Photonic Radius (Create Pin Form)</i>

Examples

```
envGetVal("layout" "photonicPinRadius")  
envSetVal("layout" "photonicPinRadius" 'float 2)
```

Related Topics

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

photonicPinRadiusLinePacket

`graphic photonicPinRadiusLinePacket string packetName`

Description

Specifies the name of the packet in which the photonic pin radius line is drawn.

GUI Equivalent

None

Examples

```
envGetVal("graphic" "photonicPinRadiusLinePacket")
envSetVal("graphic" "photonicPinRadiusLinePacket" 'string "phoPinRadiusLine")
```

Related Topics

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

photonicPinWidthLinePacket

`graphic photonicPinWidthLinePacket string packetName`

Description

Specifies the name of the packet in which the photonic pin width line is drawn.

GUI Equivalent

None

Examples

```
envGetVal("graphic" "photonicPinWidthLinePacket")
envSetVal("graphic" "photonicPinWidthLinePacket" 'string "phoPinWidthLine")
```

Related Topics

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

srcOpticalElectricalConnection

```
schematic srcOpticalElectricalConnection cyclic { "ignored" "warning" "error" }
```

Description

Specifies the severity of the schematic rules checker (SRC) check for any optical to electrical connections.

The default is `error`.

GUI Equivalent

Command	<i>Check – Rules Setup</i>
Form Field	<i>Electrical Connections (Optical tab) (<u>Schematic Rules Checks Setup</u> form)</i>

Examples

```
envGetVal("schematic" "srcOpticalElectricalConnection")  
envSetVal("schematic" "srcOpticalElectricalConnection" 'cyclic "ignored")
```

Related Topics

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

srcOpticalMultiToSingle

```
schematic srcOpticalMultiToSingle cyclic { "ignored" "warning" "error" }
```

Description

Specifies the severity of the schematic rules checker (SRC) check for any multi mode optical to single mode optical connections.

The default is `error`.

GUI Equivalent

Command	<i>Check – Rules Setup</i>
Form Field	<i>Multi Mode to Single Mode (Optical tab) (<u>Schematic Rules Checks Setup</u> form)</i>

Examples

```
envGetVal("schematic" "srcOpticalMultiToSingle")  
envSetVal("schematic" "srcOpticalMultiToSingle" 'cyclic "ignored")
```

Related Topics

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

srcOpticalSingleToMulti

```
schematic srcOpticalSingleToMulti cyclic { "ignored" "warning" "error" }
```

Description

Specifies the severity of the schematic rules checker (SRC) check for single mode nets or outputs that are connected to multi mode inputs or nets.

The default is `ignored`.

GUI Equivalent

Command	<i>Check – Rules Setup</i>
Form Field	<i>Single Mode to Multi Mode (Optical tab) (<u>Schematic Rules Checks Setup</u> form)</i>

Examples

```
envGetVal("schematic" "srcOpticalSingleToMulti")  
envSetVal("schematic" "srcOpticalSingleToMulti" 'cyclic "ignored")
```

Related Topics

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

srcOpticalTooManyConnections

```
schematic srcOpticalTooManyConnections cyclic { "ignored" "warning" "error" }
```

Description

Specifies the severity of the schematic rules checker (SRC) check for optical nets to ensure each has a maximum of 2 connections.

The default is `error`.

GUI Equivalent

Command	<i>Check – Rules Setup</i>
Form Field	<i>Too Many Connections (Optical tab) (<u>Schematic Rules Checks Setup</u> form)</i>

Examples

```
envGetVal("schematic" "srcOpticalTooManyConnections")  
envSetVal("schematic" "srcOpticalTooManyConnections" 'cyclic "ignored")
```

Related Topics

[List of Photonics Solution Environment Variables](#)

[Setting Environment Variables](#)

Forms

This section lists and describes the Virtuoso® Photonics Solution forms.

Virtuoso Photonics Solution Forms

Composite Waveguide Editor Form

Generate Selected From Layout Form

Composite Waveguide Editor Form

Use the Composite Waveguide Editor to edit a composite waveguide. A composite waveguide is a general placeholder for waveguide paths between optical components and can be generated in the layout when using the Virtuoso Photonics Solution. Using the Composite Waveguide Editor, you can edit the different building blocks of a waveguide path, such as straight waveguide and bend waveguide.

Elements

Lists the various waveguide instances that together comprise the composite waveguide.

For each waveguide element, the corresponding *Library*, *Cell*, and *View* information is also listed.

For the selected waveguide element, the adjoining preview pane displays an image of the composite waveguide, highlighting the part represented by the selected element.

Parameters

Lists the CDF parameters corresponding to the selected waveguide element. Only those parameters that correspond to the selected waveguide element are displayed. The *Orientation* parameter is applicable to all the supported waveguide elements. Therefore, the *Orientation* parameter is always displayed.

Orientation lists the supported orientations for generating the selected waveguide element.

waveguide layer specifies the valid layer on which the optical connection is supported.

Rotation specifies the angle at which the selected waveguide element can rotate.

Radius specifies the curvature of the selected waveguide element.

Length specifies the length of a straight waveguide element.

Width specifies the width of the selected waveguide element.

Total Angle (deg) specifies the length of an arc (in degrees) for the circle whose radius is specified by the *Radius* parameter.

Note: The *Total Angle (deg)* and *Total Angle (rad)* parameters are used for bend waveguides.

Total Angle (Rad) specifies the length of an arc (in radians) for the circle whose radius is specified by the *Radius* parameter. This parameter is automatically calculated using the *Total Angle (deg)* value.

Note: The *Total Angle (rad)* and *Total Angle (deg)* parameters are used for bend

Virtuoso Photonics Solution Guide

Forms

waveguide elements.

Options

Align elements rotates the subsequent components inside a composite waveguide to maintain their alignment when the selected component is edited to change its rotation, total angle, or radius.

Environment variable:

Match widths updates the width of components inside a composite waveguide to ensure the width of each subsequent component matches that of the edited waveguide component.

Environment variable: cweMatchWidths

Show updates on canvas enables the dynamic zooming in and panning on clicking *Apply* to show the updates to the actual instance on the layout canvas.

Apply zooms into and pans on the layout canvas to display the actual waveguide element selected for editing. The zoom and pan functionality is controlled by the dynamic zoom and pan functionality supported by the Display Options form.

By default, the selected waveguide element is panned into and displayed on the canvas, keeping the zoom settings unchanged. If the selected instance is currently visible, no panning or zooming is applied.



Related Topics

Generating Optical Pins

Virtuoso Photonics Solution Forms





Generate Selected From Layout Form

Use the *Generate Selected From Layout* form to customize the placement of back annotated unbound layout instances in the schematic.

Field	Description
<i>Generate</i>	<p>Specifies how instances are backannotated to the schematic.</p> <ul style="list-style-type: none"> ■ <i>individually</i>: Each instance is backannotated individually. This is the default. ■ <i>mfactored</i>: Instances are backannotated as an mfactor device in the schematic if they have the same master and connectivity, the mfactor property is also updated. ■ <i>iterated</i>: Instances are backannotated as a vectored instance if they have the same master and connectivity. <p>Environment Variable: <u>backAnnotateInstances</u></p>
<i>Rows, Columns</i>	Specifies the number of rows and columns of unbound instances for placement into the schematic.
<i>X, Y Spacing</i>	Specifies the horizontal and vertical spacing.
	Updates the layout of instances to a grid with as close to a square aspect ratio as possible.
	Switches between column-priority packing and row-priority packing for the grid of instances being placed down.
<p>The following columns in the table provide information about all dummies that are not yet backannotated to the schematic. You can select and deselect instances in this table and edit the library, cell, and view information.</p>	
<i>Name</i>	Name of the instance.
<i>Schematic Lib</i>	Name of the schematic library.
<i>Schematic Cell</i>	Name of the schematic cell.
<i>Schematic View</i>	Name of the schematic view.
<i>Type</i>	Specifies the figure type. Valid values are instances or pin.
<i>Update Master</i>	Updates the master cellview based on the information in this form.

Virtuoso Photonics Solution Guide

Forms

Field	Description
	Rotates the instance through 90 degrees clockwise.
	Rotates the instance through 90 degrees counterclockwise.
	Mirrors the instance about its y axis.
	Mirrors the instance about its x axis.

Related Topics

[Generating Selected From Layout](#)

[Generating Optical Pins](#)

Virtuoso Photonics Solution Guide

Forms
