

# **Part Developer User Guide**

**Product Version 23.1**

**September 2023**

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

Portions © Apache Software Foundation, Sun Microsystems, Free Software Foundation, Inc., Regents of the University of California, Massachusetts Institute of Technology, University of Florida . Used by permission. Printed in the United States of America.

Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA

Allegro Part Developer contains technology licensed from, and copyrighted by: Apache Software Foundation, 1901 Munsey Drive Forest Hill, MD 21050, USA © 2000-2005, Apache Software Foundation. Sun Microsystems, 4150 Network Circle, Santa Clara, CA 95054 USA © 1994-2007, Sun Microsystems, Inc. Free Software Foundation, 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA © 1989, 1991, Free Software Foundation, Inc. Regents of the University of California, Sun Microsystems, Inc., Scriptics Corporation, © 2001, Regents of the University of California. Daniel Stenberg, © 1996 - 2006, Daniel Stenberg. UMFPACK © 2005, Timothy A. Davis, University of Florida, (davis@cise.ulv.edu). Ken Martin, Will Schroeder, Bill Lorensen © 1993-2002, Ken Martin, Will Schroeder, Bill Lorensen. Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, Massachusetts, USA © 2003, the Board of Trustees of Massachusetts Institute of Technology. All rights reserved.

**Trademarks:** Trademarks and service marks of Cadence Design Systems, Inc. (Cadence) 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. Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission. 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. The information contained herein is the proprietary and confidential information of Cadence or its licensors, and is supplied subject to, and may be used only by Cadence's customer in accordance with, a written agreement between Cadence and its customer. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information. Cadence is committed to using respectful language in our code and communications. We are also active in the removal and/or replacement of inappropriate language from existing content. This product documentation may however contain material that is no longer considered appropriate but still reflects long-standing industry terminology. Such content will be addressed at a time when the related software can be updated without end-user impact.

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

# Contents

---

<u>Preface</u> .....	19
<u>Typographical Conventions</u> .....	20
<u>Related Documentation</u> .....	21
<u>1</u>	
<u>Getting Started</u> .....	23
<u>Launching Part Developer</u> .....	24
<u>2</u>	
<u>Part Developer User Interface</u> .....	25
<u>Part Developer Work Environment</u> .....	25
<u>Part Developer Menus</u> .....	27
<u>File</u> .....	27
<u>Edit</u> .....	28
<u>View</u> .....	29
<u>Tools</u> .....	29
<u>Templates</u> .....	30
<u>Graphic Editor</u> .....	31
<u>3</u>	
<u>Cell Editor</u> .....	39
<u>Cell Editor User Interface</u> .....	40
<u>Cell Editor Tree</u> .....	41
<u>Viewing Relationships</u> .....	42
<u>Package Editor</u> .....	44
<u>General</u> .....	45
<u>Package Pin</u> .....	48
<u>Part Table</u> .....	57
<u>Symbol Editor</u> .....	57
<u>General</u> .....	58

<u>Symbol Pins</u> .....	60
<u>Find</u> .....	64
<u>Symbol Editor Canvas</u> .....	65
<u>VHDL Map File Editor</u> .....	67
<u>General</u> .....	68
<u>Mapping</u> .....	69
<u>VHDL Wrapper File Editor</u> .....	71
<u>General</u> .....	72
<u>Mapping</u> .....	73
<u>Verilog Map File Editor</u> .....	74
<u>General</u> .....	74
<u>Mapping</u> .....	75
<u>Verilog Wrapper File Editor</u> .....	76
<u>General</u> .....	77
<u>Mapping</u> .....	77
<u>Row/Column Shown/Hidden Indicator</u> .....	79

## 4

<u>Configuring Part Developer</u> .....	81
<u>Setup</u> .....	81
<u>Understanding the Setup Options Tree</u> .....	82
<u>Setting Up Defaults for Low-Asserted Pins and Split Parts</u> .....	83
<u>Setting Up Package Defaults</u> .....	86
<u>Setting Up Package Pin Properties</u> .....	87
<u>Setting Up Symbol Properties</u> .....	93
<u>Setting Up Symbol Pin Defaults</u> .....	96
<u>Setting Up Symbol Pin Properties</u> .....	97
<u>Setting Up PTF Defaults</u> .....	98
<u>Setting Up Shape Defaults</u> .....	98
<u>Specifying Fonts</u> .....	99
<u>Configuration</u> .....	101
<u>Creating New Configuration</u> .....	101
<u>Using Saved Configuration to Create Parts</u> .....	103
<u>Verifying a Part Against a Template</u> .....	103
<u>Extracting Configuration from Existing Parts</u> .....	104

### 5

<u>Creating Parts</u> .....	107
<u>Part Types</u> .....	107
<u>Symmetrical Parts</u> .....	107
<u>Asymmetrical Parts</u> .....	108
<u>Split Parts</u> .....	108
<u>Part Creation Methodology</u> .....	109
<u>Creating New Cells</u> .....	110
<u>Adding Logical Pins</u> .....	112
<u>Using the Add Pin Dialog Box</u> .....	112
<u>Directly to a Package through the Package Editor</u> .....	117
<u>Directly to a Symbol through the Symbol Editor</u> .....	119
<u>Adding Global Pins</u> .....	120
<u>Setting and Retrieving Pin Order</u> .....	121
<u>Setting Pin Order</u> .....	121
<u>Retrieving Pin Order</u> .....	121
<u>Creating Packages</u> .....	121
<u>Using the Package Editor</u> .....	122
<u>Direct Generation from a Symbol</u> .....	122
<u>Entering Package Information</u> .....	123
<u>Specifying PIN_DELAY</u> .....	126
<u>Creating Split Parts</u> .....	127
<u>Adding Package Pin Properties</u> .....	128
<u>Adding Differential Pair Properties</u> .....	128
<u>Autocreating Differential Pairs through the Package Editor</u> .....	129
<u>Specifying a Naming Convention for Autocreation of Differential Pairs</u> .....	130
<u>Naming Differential Pairs</u> .....	131
<u>Creating a Differential Pair from Selected Pins</u> .....	132
<u>Deleting Differential Pair Properties</u> .....	132
<u>Some Quick Pin Mapping Techniques</u> .....	133
<u>Creating Symbols</u> .....	134
<u>Using the Symbol Editor</u> .....	135
<u>Direct Generation from a Package</u> .....	135
<u>Entering Symbol Information</u> .....	135
<u>Adding Symbol Pin Properties</u> .....	136

<u>Adding Symbol Pin Properties for Individual Bits of a Vector Pin</u>	137
<u>Adding Images to Symbols</u>	137
<u>Adding Pins in Additional Packages and Symbols</u>	138
<u>Creating Sizeable and HAS FIXED SIZE Symbols</u>	138
<u>Creating Parts with Bubble Group and Pass-Through Pins</u>	141
<u>Bubble Group and Pass-Through Pins</u>	141
<u>Creating Bubble Groups</u>	141
<u>Creating Pass-Through Pins</u>	143

## 6

<u>Creating Parts from PDFs</u>	145
<u>Importing Pin Grid</u>	150
<u>Steps</u>	150
<u>Importing Pin Table</u>	151
<u>Steps</u>	151
<u>Right-Click Options on Paste the Data Page</u>	152

## 7

<u>Modifying Parts</u>	155
<u>Modifying Logical Pins</u>	155
<u>Renaming Pins</u>	156
<u>Other Pin Modifications</u>	157
<u>Modifying Packages</u>	159
<u>Modifying and Deleting Logical and Physical Parts</u>	160
<u>Modifying Package Properties</u>	161
<u>Modifying Footprint Information</u>	161
<u>Modifying Pin Lists and Mapping</u>	163
<u>Converting Scalar Pins to Vector</u>	167
<u>Modifying Package Pin Properties</u>	167
<u>Deleting Symbol and Package Pins</u>	168
<u>Modifying Symbols</u>	170
<u>Modifying Symbol Properties</u>	171
<u>Modifying Symbol Text</u>	171
<u>Modifying Symbol Outline</u>	171
<u>Move Pins</u>	171

<u>Modifying Symbol Pin Properties</u>	172
<u>Modification Tips</u>	173
<u>Modifying a Value for Multiple Rows or Columns</u>	173
<u>Using Filters</u>	174
<u>Using Find Filter to Locate Symbol Objects</u>	175

## 8

<u>Creating Shapes</u>	177
<u>Types of Shapes</u>	177
<u>Pin Shapes</u>	178
<u>Custom Shapes</u>	178
<u>Shape Editor</u>	179
<u>Setting Up a Project for Shapes</u>	181
<u>Creating a Shape</u>	182
<u>Extracting Shapes from Existing Symbols</u>	183
<u>Replacing Symbol Pin Shapes</u>	183
<u>Attaching Custom Shapes to Existing Symbol Pins</u>	184
<u>Inserting Custom Shapes</u>	185
<u>Aligning Custom Shapes</u>	185
.....	186

## 9

<u>Working with VHDL Wrappers and Map Files</u>	187
<u>Creating a VHDL Map File</u>	189
<u>Creating a VHDL Wrapper File</u>	195
<u>Modifying a VHDL Wrapper/Map File</u>	200
<u>Deleting a VHDL Wrapper/Map File</u>	200
<u>Renaming a VHDL Wrapper/ Map File</u>	200

## 10

<u>Creating Verilog Wrappers and Map Files</u>	201
<u>Creating a Verilog Map File</u>	201
<u>Creating a Verilog Wrapper File</u>	207
<u>Modifying a Verilog Wrapper/Map File</u>	211

<u>Deleting a Verilog Wrapper/Map File</u>	211
<u>Renaming a Verilog Wrapper/Map File</u>	211

## 11

<u>Symbol Property Templates</u>	213
<u>Components of a Symbol Property Template</u>	213
<u>Creating a Symbol Property Template</u>	214
<u>Opening a Symbol Property Template</u>	217
<u>Applying a Symbol Property Template</u>	218
<u>Applying a Symbol Property Template to a Symbol</u>	219
<u>Applying a Symbol Property Template to All Symbols</u>	219
<u>Applying a Symbol Property Template to the Setup</u>	220
<u>Extracting a Symbol Property Template from an Existing Symbol</u>	220

## 12

<u>Import and Export</u>	223
<u>Import Export Methodology</u>	224
<u>Methodology Used When Updating Parts Using the ECO Process</u>	225
<u>Example</u>	230
<u>Import APD Component Files</u>	233
<u>Conversion Details</u>	233
<u>Steps</u>	233
<u>Import Capture Part (Windows Only)</u>	234
<u>Conversion Details</u>	235
<u>Steps</u>	239
<u>Importing a Capture Library</u>	240
<u>Import EDAXML Part</u>	240
<u>Post Import Issues</u>	241
<u>Import Si2 PinPak XML Part</u>	243
<u>Conversion Details</u>	243
<u>Steps</u>	245
<u>Import Comma Separated Value (.csv) File</u>	246
<u>Conversion Details</u>	246
<u>CSV Import Example</u>	252
<u>Steps</u>	254

## Part Developer User Guide

---

<u>Import Synopsys PTM Model</u>	255
<u>Conversion Details</u>	255
<u>Example</u>	256
<u>Steps</u>	256
<u>Import Verilog Model</u>	257
<u>Conversion Details</u>	257
<u>Steps</u>	257
<u>Import VHDL Model</u>	258
<u>Conversion Details</u>	258
<u>Steps</u>	259
<u>Import FPGA</u>	259
<u>Steps</u>	260
<u>Location of Actel FPGA Libraries</u>	264
<u>Import Text File</u>	264
<u>Conversion Details</u>	264
<u>Profile Use Model</u>	269
<u>Steps</u>	271
<u>Import ViewLogic(VL) Part</u>	273
<u>Conversion Details</u>	273
<u>Steps</u>	276
<u>Import Allegro Footprint</u>	277
<u>Conversion Details</u>	277
<u>Steps</u>	277
<u>Import Die Text</u>	278
<u>Conversion Details</u>	278
<u>Steps</u>	278
<u>Import DML Model</u>	279
<u>Conversion Details</u>	279
<u>Steps</u>	280
<u>Import IBIS Model</u>	280
<u>Conversion Details</u>	280
<u>Steps</u>	281
<u>Import Mentor Part</u>	282
<u>Conversion Details</u>	282
<u>Steps</u>	284
<u>Import Pin Grid</u>	285

## Part Developer User Guide

---

<u>Import Pin Table</u>	285
<u>Import ECO - APD Component Files</u>	285
<u>Conversion Details</u>	285
<u>Steps</u>	285
<u>Import ECO - Capture Part (Windows Only)</u>	286
<u>Steps</u>	286
<u>Import ECO - EDAXML Part</u>	287
<u>Steps</u>	287
<u>Import ECO - Si2 PinPak XML Part</u>	288
<u>Steps</u>	288
<u>Import ECO - Comma Separated Value (.csv) File</u>	289
<u>Steps</u>	289
<u>Import ECO - Synopsis PTM</u>	289
<u>Steps</u>	290
<u>Import ECO - FPGA</u>	290
<u>Steps</u>	290
<u>Import ECO - Text File</u>	291
<u>Steps</u>	291
<u>Import ECO - ViewLogic(VL) Part</u>	292
<u>Steps</u>	293
<u>Import ECO - Allegro Footprint</u>	293
<u>Steps</u>	293
<u>Import ECO - Die Text</u>	294
<u>Steps</u>	294
<u>Import ECO - DML Model</u>	295
<u>Steps</u>	295
<u>Import ECO - IBIS Model</u>	296
<u>Steps</u>	296
<u>Import ECO - Mentor Part</u>	297
<u>Steps</u>	297
<u>Import ECO - Pin Grid</u>	297
<u>Steps</u>	298
<u>Import ECO - Pin Table</u>	298
<u>Steps</u>	298
<u>Export Capture Part (Windows Only)</u>	299
<u>Steps</u>	299

## Part Developer User Guide

---

<u>Conversion Details</u>	300
<u>Translation of the Chips (Physical) View</u>	300
<u>Export EDAXML Part</u>	304
<u>Steps</u>	305
<u>Conversion Details</u>	305
<u>Export Comma Separated Value (.csv) File</u>	309
<u>Steps</u>	309
<u>Export ViewLogic(VL) Part</u>	310
<u>Steps</u>	310
<u>Export Mentor Part</u>	310
<u>Steps</u>	311
 <b>13</b>	
<b>Verifying Parts</b>	313
<u>View Verification</u>	313
<u>Instantiation and Packaging</u>	314
<u>con2con Instantiation and Packaging Checks</u>	314
<u>hlibftb Instantiation and Packaging Checks</u>	315
<u>Advanced View Checks</u>	316
<u>VHDL Compilation</u>	316
<u>Verilog Compilation</u>	316
<u>Verify with Templates</u>	317
<u>Property Checks</u>	317
<u>Pin Load Checks</u>	317
<u>Symbol Checks</u>	317
 <b>14</b>	
<b>Interface Comparator</b>	321
<u>Steps to Run Interface Comparator</u>	321
<u>Points to Remember when Running Interface Comparator</u>	322
 <b>15</b>	
<b>SI Model Interface Comparison</b>	329
<u>To run the SI Model Interface Comparison:</u>	329

<u>SI Model Interface Comparison Results</u>	329
<b>16</b>	
<u>Part Logging and Versioning</u>	331
<u>Revision Editor</u>	332
<u>Starting Part Logging and Versioning</u>	334
<u>Viewing the Revision Log</u>	335
<u>Adding Your Comments to the Revision Log</u>	336
<u>Stopping Part Logging and Versioning</u>	337
<u>Restarting the Part Logging and Versioning</u>	337
<u>Modifications that Result in Major and Minor Number Changes</u>	337
<b>17</b>	
<u>Advanced Tasks</u>	341
<u>Creating Personal Configuration Files through the CDS_SITE Environment Variable</u>	341
<u>Cleaning the Part Developer Registry Entry</u>	342
<u>Configuring to Accept ? as the Only Placeholder Property Value</u>	342
<u>Translating Pin Types on Copy-Pasting from PDFs and Importing Data from CSV Files</u>	343
<u>Specifying Default Pin Types for Text Import</u>	344
<u>Specifying Default Pin Types for Allegro Footprint Import</u>	344
<u>Adding External Tools to Part Developer</u>	344
<u>Example 1</u>	346
<u>Example 2</u>	346
<u>Modifying the Sheet Size</u>	347
<u>Modifying RefDes Prefix and Class Values</u>	348
<u>Modifying the Default List of Package, Package Pin, Symbol, and Symbol Pin Properties</u>	349
<u>Configuring Translation Rules for Import and Export</u>	351
<u>Configuring Mentor Export</u>	355
<u>Configuring the Predefined Headers for CSV Import</u>	357
<u>Configuring the Predefined Headers for Text Import</u>	359
<u>Configuring Pin Text Position Defaults</u>	360
<u>Configuring to Use Square Brackets in Vector Pin Names</u>	360
<u>Importing Pins with Square Brackets in Basenames</u>	361
<u>Configuring the Alignment of Symbol Properties</u>	361

<u>Configuring the Alignment of Symbol Pin Properties</u>	362
<u>Configuring the Rotation of Symbol Pin Properties</u>	362
<u>Configuring the Placement and Stackup of Symbol Properties</u>	363

## 18

<u>Part Developer Console Commands</u>	365
<u>csv2ptf</u>	365
<u>csv2ptf Process Overview</u>	365
<u>Configuring Conversion Rules for csv2ptf Import</u>	366
<u>Creating a Configuration File for csv2ptf Import</u>	366
<u>Usage</u>	367
<u>Example</u>	368
<u>con2con</u>	370
<u>Usage</u>	370
<u>Example</u>	373
<u>viewlogic2con</u>	373
<u>Usage</u>	373
<u>Example</u>	375
<u>xml2con</u>	375
<u>Usage</u>	375
<u>Example</u>	376
<u>csv2con</u>	376
<u>Usage</u>	377
<u>Example</u>	378
<u>pinpak2con</u>	378
<u>Usage</u>	378
<u>Example</u>	380
<u>ptm2con</u>	380
<u>Usage</u>	380
<u>Example</u>	381
<u>vhdl2con</u>	382
<u>Usage</u>	382
<u>Example</u>	383
<u>verilog2con</u>	383
<u>Usage</u>	383

## Part Developer User Guide

---

<u>Example</u>	384
<u>con2xml</u>	384
<u>Usage</u>	384
<u>Example</u>	385
<u>con2csv</u>	385
<u>Usage</u>	385
<u>Example</u>	386
<u>cap2con</u>	386
<u>Usage</u>	387
<u>Example</u>	388
<u>con2cap</u>	388
<u>Usage</u>	388
<u>Example</u>	389
<u>cap2xml</u>	390
<u>Usage</u>	390
<u>Example</u>	390
<u>xml2cap</u>	391
<u>Usage</u>	391
<u>Example</u>	391
<u>apd2con</u>	392
<u>Usage</u>	392
<u>Example</u>	393
<u>dml2con</u>	393
<u>Usage</u>	393
<u>Example</u>	395
<u>ibis2con</u>	395
<u>Usage</u>	396
<u>Example</u>	397
<u>lib2cellptf</u>	398
<u>Usage</u>	398
<u>Example</u>	399
<u>fpga2con</u>	399
<u>Usage</u>	399
<u>Example</u>	402
<u>allegro2con</u>	402
<u>Usage</u>	402

<u>Example</u>	403
<u>text2con</u>	404
<u>Usage</u>	404
<u>Example</u>	405

## A

<u>Dialog Box Help</u>	407
<u>Open Project</u>	407
<u>New Cell</u>	407
<u>Open Cell</u>	408
<u>Edit Functions</u>	408
<u>Distribute Pins</u>	409
<u>Add Pin</u>	410
<u>Rename</u>	415
<u>Extracted Shape Name</u>	416
<u>Rename Pin</u>	416
<u>Delete Package Pin</u>	416
<u>Add Properties</u>	417
<u>Rename Package Pin Property</u>	417
<u>Delete Package Pin Property</u>	418
<u>Add Physical Pin Numbers</u>	418
<u>Delete Symbol Pin</u>	419
<u>Symbol Pin Attributes</u>	419
<u>Map Pin Name and Text</u>	421
<u>Rename Symbol Pin Property</u>	422
<u>Delete Symbol Pin Property</u>	422
<u>Move Pin</u>	423
<u>Select Package</u>	423
<u>Modify Package</u>	424
<u>Select Model</u>	425
<u>Modify Model</u>	425
<u>Lib:Cell:View – Select Model</u>	425
<u>Import and Export Wizard</u>	426
<u>Select Source</u>	426
<u>Select Destination</u>	426

## Part Developer User Guide

---

<u>Select Capture Part</u>	426
<u>Preview of Import Data</u>	427
<u>Select Package and Symbol(s)</u>	428
<u>Select Package</u>	429
<u>Select Package</u>	429
<u>Select Package</u>	429
<u>Select Associated Package(s) or Unassociated Symbol(s)</u>	429
<u>ECO Messages</u>	429
<u>Paste the data</u>	431
<u>Preview of Derived Data</u>	431
<u>Duplicate Pin Resolver</u>	432
<u>Select Rows</u>	432
<u>Select Delimiter(s)</u>	433
<u>Select Columns</u>	434
<u>Select Views</u>	435
<u>Select Footprint</u>	435
<u>Select Footprint</u>	436
<u>Location</u>	436
<u>Name</u>	436
<u>Browse Jedec Type</u>	436
<u>Browse Alt Symbol</u>	436
<u>Save All</u>	437
<u>Shapes – Save All</u>	437
<u>Annotate Generics</u>	438
<u>Annotate Parameters</u>	439
<u>Select Value to be Annotated</u>	440
<u>Add Logical Part</u>	440
<u>Add Physical Part</u>	440
<u>Rename Physical Part</u>	441
<u>Symbol Pin Property Attributes</u>	441
<u>Save As</u>	442
<u>Modify Properties</u>	442
<u>Default Model</u>	443
<u>UPPER CASE Property</u>	443
<u>Generate Symbol(s)</u>	443
<u>Modify Pin</u>	444

## Part Developer User Guide

---

<u>Filter Rows</u>	.....	446
<u>Edit Global Mapping</u>	.....	446
<u>Modify Column Values</u>	.....	447
<u>Select Package to Associate</u>	.....	447
<u>Specify The Symbol Size</u>	.....	448
<u>Find &amp; Replace</u>	.....	448
<u>SI Model Interface Comparison</u>	.....	449
<u>SI Model Interface Comparison Results</u>	.....	449
<u>New Shape</u>	.....	451
<u>Custom</u>	.....	451
<u>Pin</u>	.....	451
<u>Open Shape</u>	.....	451
<u>Custom</u>	.....	451
<u>Pin</u>	.....	451
<u>Text Properties</u>	.....	452
<u>Template Application Wizard</u>	.....	452
<u>Cadence Product Choices</u>	.....	453

## B

<u>Errors and Warnings</u>	.....	457
<u>Errors and Warnings</u>	.....	458

## C

<u>Shortcut Menu Options</u>	.....	551
<u>Shortcut Menu Options from Cell Tree Nodes</u>	.....	551
<u>Packages</u>	.....	551
<u>Package Names</u>	.....	551
<u>Symbols</u>	.....	552
<u>Symbol Names (sym_n)</u>	.....	552
<u>Part Table Files</u>	.....	553
<u>Part Table File Names (&lt;name&gt;.ptf)</u>	.....	553
<u>VHDL/Verilog MapFiles</u>	.....	554
<u>VHDL/Verilog mapfile names</u>	.....	554
<u>Primitive Names in VHDL/Verilog Mapfile Names</u>	.....	555
<u>VHDL/Verilog Wrappers</u>	.....	556

## Part Developer User Guide

---

<u>VHDL/Verilog map file names</u>	556
<u>Model Names in VHDL/Verilog Wrappers</u>	556
<u>Menu Options in Logical/Physical/Global Pins and Properties Tables</u>	557
<u>Grid Usability Tips</u>	558
 <b>D</b>	
<u>List of Valid Values in Part Developer</u>	561
<u>Supported Characters in Cell Names</u>	561
<u>Unsupported Characters for Cell Names</u>	561
<u>Unsupported Characters for Property Values in PTF File</u>	562
<u>Unsupported Characters for PART_NAME Property Values</u>	562
<u>Supported Characters for PACK_TYPE Property Values</u>	563
<u>Supported Characters for Property Names in Packages/Symbols/Package Pins/Symbol Pins</u>	563
<u>Unsupported Characters for Packages/Symbols/Package Pins/Symbol Pin Property Values</u>	563
<u>Supported Characters for Differential Pair Names</u>	564
<u>Pin Naming</u>	564
<u>Pin Name/Number Length</u>	566
<u>Primitive Name Length</u>	566
<u>Body Section Property Name Length</u>	566
<u>Body Section Property Value Length</u>	566
<u>PORT_ORDER Value Length</u>	566
<u>Valid Range Notations</u>	566
<u>NMP</u>	567
 <b>E</b>	
<u>Pin Types</u>	569
<u>Index</u>	573

# Preface

---

Part Developer and Library Explorer functionality is divided into two levels. The core capabilities are found in all versions of Library Explorer and Part Developer and shipped as part of PCB Librarian and the latest packaging configurations of Allegro Design Entry HDL. The extended capabilities are restricted only to the Allegro Managed Library Authoring license.

Besides creating library parts in a quick and intuitive manner, you can also utilize information provided over the Internet to auto-generate the library parts. Part templates, which reflect your companies' library-creation specifications, ensure consistency across all created parts. The capability to read and write both Design Entry HDL and OrCAD Capture parts is especially helpful when leveraging existing Design Entry HDL or OrCAD libraries to drive new part creation. The support for CSV, Verilog, VHDL, and XML standards provide a way to automatically import pin and package data. The support for XML standards also provides a universal format in which to store and transfer part data.

This guide assumes familiarity with a system text editor, HDL language concepts, and the following Cadence tools used to create component symbols and models:

- Library Explorer, which lets you manage component libraries
- Part Table Editor, which lets you create part table files
- Allegro Design Entry HDL, which lets you create logic designs by drawing schematics using symbols and functional blocks
- Packager-XL, which lets you prepare your schematic for PCB layout
- Allegro PCB Editor, which lets you create and manage physical layouts

## Features Available with Allegro Managed Library Authoring License

The following features are available only in the Allegro Managed Library Authoring license:

- Map/wrapper file creation and modification
- Symbol property templates
- All types of import and export
- Interface Comparator

- Part logging and versioning
- Symbol Editor
- Shape Editor
- Footprint Viewer
- Selection and extraction from a footprint
- Verification with all footprints
- Selection and verification with footprints
- con2con instantiation and packaging checks

## Typographical Conventions

This list describes the syntax conventions used for tools used in the library development and management process. Where applicable, exceptions to these conventions are explicitly indicated.

---

literal (LITERAL)	Nonitalic or (UPPERCASE) words indicate key words that you must enter literally. These keywords represent command (function, routine) or option names.
argument	Words in italics indicate user-defined arguments for which you must substitute a value.
	Vertical bars (OR bars) separate possible choices for a single argument. They take precedence over any other character.  For example, command <i>argument</i>   <i>argument</i>
[]	Brackets denote optional arguments. When used with OR bars, they enclose a list of choices. You can choose one argument from the list.
{ }	Braces are used with OR bars and enclose a list of choices. You must choose one argument from the list.

## Part Developer User Guide

### Preface

---

...	An ellipsis indicates that you can repeat the previous argument. If they are used with brackets, you can specify zero or more arguments. If they are used without brackets, you must specify at least one argument, but you can specify more.
	<i>argument....</i> : specify at least one argument, but more are possible
	[ <i>argument</i> ]....: you can specify zero or more arguments
,...	A comma followed by an ellipsis indicates that if you specify more than one argument, you must separate those arguments by commas.
Courier font	Text in Courier font indicates command-line examples.

## Related Documentation

The following manuals give you information about other tools used during the library and part creation and management process:

If you want to know...	Read
How to manage digital component libraries	Library Explorer User Guide
How to create Part Table Files	Part Table Editor User Guide
How to use Design Entry HDL to enter schematics	Allegro Design Entry HDL User Guide
More about Allegro Design Entry HDL digital libraries	Allegro Design Entry HDL Libraries Reference
More about how to create and use physical layouts	Allegro PCB Editor documentation
More about properties supported by Cadence PCB design software	PCB and IC Packaging Properties Reference

## **Part Developer User Guide**

### Preface

---

---

#### **If you want to know...    Read**

---

More about map and    Allegro Design Entry HDL Libraries Reference  
wrapper files

More about verification    Allegro Design Entry HDL Libraries Reference  
utilities

---

---

# Getting Started

---

Allegro platform provide you a set of tools to perform library creation and management tasks. Depending upon whether you are a librarian or a designer, you can use these tools to perform specific tasks.

**Table 1-1 Tasks performed by a Librarian**

<b>Task</b>	<b>Tool Used</b>
Creating or accessing a build area.	Library Explorer
Importing reference libraries or parts to be modified into the build area.	Library Explorer
Creating new libraries or cells.	Library Explorer
Verifying libraries.	Library Explorer
Exporting new or modified libraries and parts to the reference library location.	Library Explorer
Clean up the build library area when finished.	Library Explorer
Creating and modifying symbols, packages, simulation and part table views.	Part Developer
Verifying a part.	Part Developer
Creating, modifying, and verifying a part table.	Part Table Editor
Adding new parts to a part table.	Part Table Editor

**Table 1-2 Tasks Performed by a Designer**

S.No	Task
1.	Creating a project using Project Manager.
2.	Specify the libraries to be used in the project.  <b>Note:</b> Use Part Developer to create a new part or modify existing parts in the project libraries. When you launch the Part Developer tool, it will display the project libraries, thus enabling you to modify and create new parts only in them.

**Note:** Library Explorer is not be available on a project created through Project Manager as library management tasks, such as creating new libraries and categories and copying libraries, should be done only by a librarian.

## Launching Part Developer

Part Developer can be launched from the following:

- Project Manager
  - a. Choose *Tools – Library Tools – Part Developer*.
- Command prompt
  - Type pdv
- Library Explorer
  - a. Select the part that you want to edit.
  - b. Select *Tools – Part Developer*.

## **Part Developer User Interface**

---

Part Developer is a tool for creating, editing, and verifying part data. It can be launched directly from the command line, Project Manager, or Library Explorer. Using Part Developer, you can edit the schematic symbols, physical pin data, and part table data and create simulation views for multiple parts — all within a single user interface and a single session. With built-in error checking, Part Developer ensures that parts are consistent and error-free. Part Developer also has the ability to handle asymmetrical, large pin-count, and technology-independent parts. Part Developer supports automatic part-data entry through various types of import and adherence to company guidelines using part templates.

This chapter covers the following:

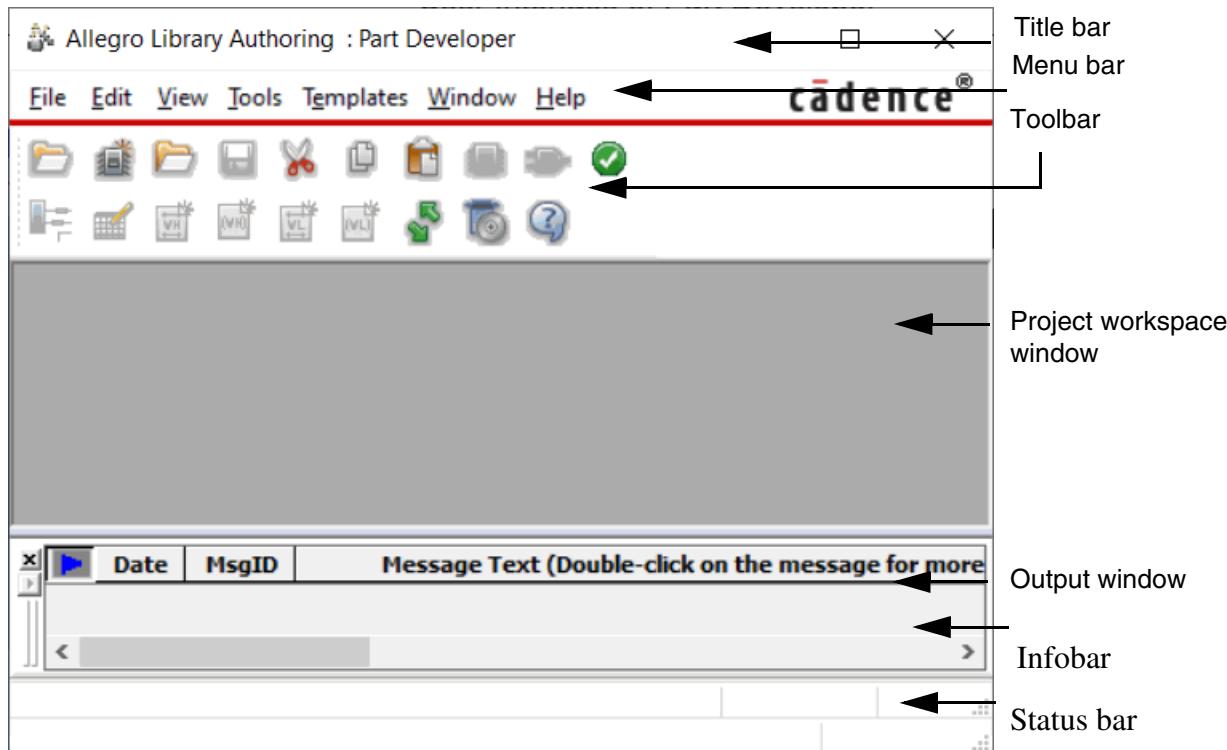
- [Part Developer Work Environment](#) on page 25
- [Part Developer Menus](#) on page 27

## **Part Developer Work Environment**

Part Developer follows the standard Windows interface model. The user interface is divided into sections as described below:

## Part Developer User Guide

### Part Developer User Interface



Window Components	Description
Title bar	Displays the name of the current project and the part that is open.
Menu bar	Provides the various options that enable you to perform all the tasks involved in creating parts.
Toolbar	Provides a single-click method to do most of the common tasks involved in creating parts.
Project Workspace Window	Displays the Cell Editor. The Cell Editor is the interface that enables you to create and manage parts. One instance of the Cell Editor is created in the project workspace window for each part that is getting created or modified. You can toggle between the different Cell Editor instances using the <i>Window</i> menu option.
Output Window	Displays the Session Log for the current session of Part Developer. All tasks performed by Part Developer are logged in the <code>session.log</code> file. The <code>session.log</code> file is saved in the project directory.

Window Components	Description
Infobar	Displays the various messages that are generated when different tasks are performed in Part Developer.
Status bar	Displays the status of Part Developer.

## Part Developer Menus

### File

The *File* menu handles the I/O operations of Part Developer. The *File* menu has the following options:

**Table 2-1 Part Developer File Menu**

Menu Option	Description
New	Creates cells, packages, symbols, Verilog and VHDL wrappers and map files, and cell-level part table files (PTF).
Open	Loads a cell into the Cell Editor
Close	Closes the currently active Cell Editor
New Project	Creates a new library or design project. You create library projects to create and manage part libraries. Design projects are created for creating schematics. To know more about creating design and library projects, see <i>Project Manager User Guide</i> and <i>Library Explorer User Guide</i> .
Open Project	Opens an existing project
Close Project	Closes the project that is loaded in Part Developer
View File – Chips	Opens the Chips.
View File – Part Table	Opens the Part Table File.
Save	Saves the project, the part, or any of the views of a part
Save As	Saves the cell with a different name in another library
Save Shape As	Saves the shape with a different name
Save All	Saves all the parts and views that are open in Part Developer

**Table 2-1 Part Developer File Menu**

Menu Option	Description
Import and Export	Creates parts from a variety of data sources, such as XML and Verilog models, and saves part data in Capture or XML format.
Change Product	Changes the license suite.
	 <b>Caution</b> <i>Changing the license suite from PCB Librarian XL to any other licence suite disables the advanced features of Part Developer.</i>
Exit	Closes Part Developer. You may be prompted to save your changes.

## Edit

The *Edit* menu handles the edit operations, such as copy, paste, and delete. The *Edit* menu provides the following options:

### Part Developer Edit Menu

Menu Option	Description
Cut	Copies a selected object to the Clipboard while removing it from the existing location. This command is available only if the source or target is appropriate.
Copy	Copies a selected object to the Clipboard without removing it from the existing location. This command is available only if the source or target is appropriate.
Paste	Places any object stored in the Clipboard into the selected location. This command is unavailable if the Clipboard is empty or if the source or target is inappropriate.  Pasting objects from the Clipboard does not affect the Clipboard contents. Use Paste to copy objects to another page or part.
Find & Replace	Launches the Find & Replace dialog box using which you can find and replace a specified string

## Part Developer User Guide

### Part Developer User Interface

---

#### Part Developer Edit Menu

Menu Option	Description
Paste Special(Grid) – Convert WhiteSpaces to NewLine	Converts the white spaces to a new line when pasting data from the Clipboard. This is useful in situations such as when pasting data from a PDF datasheet into the <i>Logical Pins</i> grid.
Paste Special(Grid) – Transpose	Converts the space-separated entries in a row of data into separate rows when pasting data from the Clipboard. For example, if the data is in the format a b c, when pasted into Part Developer, a, b, and c will appear in three separate rows. This is useful in situations such as when pasting data from a PDF datasheet into the <i>Logical Pins</i> grid.
Clear Session Log	Clears the Session Log.

---

#### View

The *View* menu controls the appearance of the toolbar and the Session Log. The *View* menu has the following options:

#### Part Developer View Menu

Menu Option	Description
Toolbars	Shows/hides the toolbars in the Part Developer UI
Session Log	Shows/hides the Session Log in the Part Developer UI

---

#### Tools

The *Tools* menu provides access to several utilities and checks that the parts are created correctly. You can also configure Part Developer by using this menu option. The *Tools* menu has the following options:

#### Part Developer Tools Menu

Menu Option	Description
Setup	Enables you to configure the setup options of Part Developer. You can configure the default values for properties, pin, symbol, format, and pin interpretation.
PCB Editor Setup	Launches the <i>User Preferences Editor</i> of PCB Editor

## Part Developer User Guide

### Part Developer User Interface

---

#### Part Developer Tools Menu

Menu Option	Description
Configuration	Provides options to create and edit part configuration and apply to setup
Verify	Runs a number of checks to ensure that the part is correct and usable in the design flow
SI Model Interface Comparison	Launches the SI Model Interface Comparison dialog box. For more information, see <a href="#">SI Model Interface Comparison</a> on page 449.
PCB Symbol Editor	Launches PCB Symbol Editor. If a particular license suite does not have the PCB Symbol Editor license, PCB_Viewer_Plus is launched. If a footprint is selected in the cell tree, PCB Symbol Editor/PCB_Viewer_Plus is launched on the selected footprint.
Design Entry HDL	Launches Design Entry HDL on the selected symbol.
Part Table Editor	Launches the Part Table Editor. The Part Table Editor is used to create a new part table file or edit an existing one.
Allegro Symbol Editor	Launches Allegro Symbol Editor on the selected symbol.
ECAD-MCAD Library Creator	Launches ECAD-MCAD Library Creator.

---

#### Templates

The *Templates* menu provides you the option to create and manage symbol property templates.

#### Part Developer Templates Menu

Menu Option	Description
New	Creates a new symbol property template
Open	Opens an existing symbol property template

---

## Graphic Editor

The *Graphic Editor* menu provides you the options to create and edit symbols and shapes. The *Graphic Editor* menu provides options to modify , Draw, Align, Nudge, Rotate, Layout, Canvas, and Font.

**Table 2-2 Part Developer Graphics Menu**

Menu Option	Description
Zoom/Pan	<p>The <i>Zoom/Pan</i> option provides a variety of magnification options. You can also pan the symbol or the shape through this option.</p> <p>The <i>FullScreen</i> and <i>Find Filter</i> options are not available for shape editing.</p>
Zoom In	Magnifies the symbol or the shape by 25 percent Accelerator Key: F11
Zoom Out	Reduces the symbol or shape size by 25 percent Accelerator Key: F12
Zoom Fit	Magnifies the symbol or the shape so as to fit the editing canvas Accelerator Key: F2
Zoom Custom	Magnifies the symbol or the shape by the specified magnification percentage
Zoom By Points	Magnifies the image by the selected number of points. The points are selected on the editing canvas by using the drag-and-select method.
Zoom By Selection	Magnifies the selected area of the symbol or shape. The selection is made on the editing canvas by using the drag-and-select method.
Pan	Enables you to pan the symbol or the shape on the editing canvas. If you are using a three-button mouse, you can pan using the middle mouse button.

**Table 2-2 Part Developer Graphics Menu**

Menu Option	Description
FullScreen	<p>Toggles between default display and full-screen display. In default display, the Part Developer window displays the tree pane, the attributes pane, and the Symbol Editor canvas. In full-screen display, the Symbol Editor canvas is displayed full-screen.</p> <p>The FullScreen option can also be accessed using the  tool button.</p>
Draw	<p>The <i>Draw</i> option provides a number of ways in which you can modify the symbol by drawing geometric figures, such as lines, polygons, and arcs.</p> <p>You can end the draw mode by pressing the <code>Esc</code> key or by right-clicking and choosing the <i>Done</i> option.</p> <p>The <i>Image</i> option is not available for shape editing.</p>
Select	<p>Enables you to make a selection on the editing canvas. If pins are selected in the Symbol Pins panel, they are automatically shown selected on the Symbol Editor canvas.</p>
Properties	<p>Launches the property editor to enable you to change the properties, such as color, line width, and line style, of a line.</p> <p><b>Accelerator Key:</b> <code>Ctrl +P</code></p>
Line	<p>Enables you to draw a line on the editing canvas</p> <p><b>Accelerator Key:</b> <code>Shift +L</code></p>
Polyline	<p>Enables you to draw a polyline on the editing canvas</p> <p><b>Accelerator Key:</b> <code>Shift +P</code></p>
Polygon	<p>Enables you to draw a polygon on the editing canvas</p> <p><b>Accelerator Key:</b> <code>Shift +G</code></p>
Rectangle	<p>Enables you to draw a rectangle on the editing canvas</p> <p><b>Accelerator Key:</b> <code>Shift +R</code></p>

## Part Developer User Guide

### Part Developer User Interface

---

**Table 2-2 Part Developer Graphics Menu**

Menu Option	Description
Arc	Enables you to draw an arc on the editing canvas Accelerator Key: Shift +A
Circle	Enables you to draw a circle on the editing canvas Accelerator Key: Shift +C
Text	Enables you to add text on the editing canvas Accelerator Key: Shift +T
Image	Enables you to add an image Accelerator Key: Shift +I The image to be added should be stored in a .bmp file.
Align	The <i>Align</i> option provides the ability to align multiple objects in relation to an anchor object. By default, the last selected object is used as the anchor object.
Top	Aligns the top margins of the selected objects with respect to the anchor object.
Middle	Aligns the selected objects to be in the middle with respect to the anchor object.
Bottom	Aligns the bottom margin of the selected objects with respect to the anchor object.
Left	Left-aligns the selected objects with respect to the anchor object.
Center	Center-aligns the selected objects with respect to the anchor object.
Right	Right-aligns the selected objects with respect to the anchor object
Nudge	The <i>Nudge</i> option enables you to move the selection by one point.
Up	Moves the selection up by one point
Down	Moves the selection down by one point
Left	Moves the selection to the left by one point

## Part Developer User Guide

### Part Developer User Interface

---

**Table 2-2 Part Developer Graphics Menu**

Menu Option	Description
Right	Moves the selection to the right by one point.
Rotate	The <i>Rotate</i> option enables you to rotate a selected object. Objects always rotate around their centers.
Left	Rotates the selection to the left
Right	Rotates the selection to the right
Flip Horizontal	Horizontally flips the selection
Flip Vertical	Vertically flips the selection
Layout	The <i>Layout</i> option provides you the ability to manipulate the layout of the symbol
Space Horizontally	Evenly spaces out the selected components horizontally.
Space Vertically	Evenly spaces out the selected components vertically
Make Same Width	Makes all the selected objects the same width.
Make Same Height	Makes all the selected objects the same height.
Make Same Size	Makes all the objects the same size.
Canvas	Enables you to manipulate the editing canvas  The Extract Shape and Replace PinShape options are not available for shape editing.
Undo	Undoes the Symbol Editor or Shape Editor changes  Accelerator Key: Ctrl +Z
Redo	Redoes the Symbol Editor or Shape Editor changes  Accelerator Key: Ctrl +Y

**Table 2-2 Part Developer Graphics Menu**

Menu Option	Description
Pin Grid	<p>Displays the default pin grid or hides all grid lines on the editing canvas</p> <p>The following tool button has the same function:</p> 
Non-Pin Grid	<p>Displays the finer grid on the editing canvas</p> <p>The following tool button has the same function</p> 
Snap to Grid	<p>Determines whether all graphical object operations, such as move and scale, take place on the grid or between grid points</p> <p>Regardless of the Snap to Grid selection, all operations on pins take place on the pin grid.</p> <p>For information on grid behavior, see <a href="#">Symbol Editor Grid Settings and Behavior</a> on page 66</p>

## Part Developer User Guide

### Part Developer User Interface

---

**Table 2-2 Part Developer Graphics Menu**

---

Menu Option	Description
Group	<p>Groups the selected objects. Grouping makes it easy to move a collection of objects in a single step as all objects are moved together and their relative distances are maintained.</p> <p>Accelerator Key: Ctrl +G</p> <p>A group can contain the following objects:</p> <ul style="list-style-type: none"><li>■ Pins</li><li>■ Basic shapes</li><li>■ Custom shapes</li><li>■ Another group</li></ul> <p>An object cannot be part of two groups unless one group is a subset of the other group.</p> <p><b>Note:</b> Although you can move, rename, scale, remove, cut, copy, or paste a group, you cannot perform pin operations on a group. To perform any pin operation, such as replacing the pin shape and changing the pin position and location, on a group, you need to first ungroup the grouped objects.</p>
Ungroup	<p>Ungroups the grouped objects</p> <p>Accelerator Key: Ctrl +U</p> <p> <i>Tip</i></p> <p>Ungrouping the objects of a nested group is a two-step process. First, you need to ungroup the group that contains the nested group and then ungroup the nested group.</p>
Front	<p>Brings the selected object to the front</p> <p>Accelerator Key: Ctrl +F</p>

## Part Developer User Guide

### Part Developer User Interface

---

**Table 2-2 Part Developer Graphics Menu**

Menu Option	Description
Back	Sends the selected object to the back Accelerator Key: Ctrl +B
Forward	Brings the selected object forward by one layer Accelerator Key: Ctrl +W
Backward	Sends the selected object backward by one layer Accelerator Key: Ctrl +K
Replace PinShape	Replaces the shape of an existing symbol pin with another shape. You can also select the objects around the pin and replace them along with the pin in a single operation. Accelerator Key: Ctrl +R
Extract Shape	Extracts a shape from an existing symbol Accelerator Key: Ctrl +T
Copy to TextShape	Accelerator Key: Ctrl +D
Paste to TextShape	Accelerator Key: Ctrl +S
Snap To Absolute Grid	With the Snap to Grid option selected, select this option to move a group of non-pin elements to a new location while maintaining the relative position of the group elements with respect to the grid.  <b>Note:</b> This command has no effect if the <i>Snap to Grid</i> option is not selected.
Font	Enables you to increase or decrease the font size of the selected text
Increase Font	Increases the font size by 2 points Accelerator Key: Ctrl + Shift + >
Decrease Font	Decreases the font size by 2 points Accelerator Key: Ctrl + Shift + <

## Part Developer User Guide

### Part Developer User Interface

---

**Table 2-2 Part Developer Graphics Menu**

---

<b>Menu Option</b>	<b>Description</b>
Additional Shortcut Keys for Font Editing	You can use the following shortcut keys to increase and decrease the font size by 1 point:  Ctrl + ] Ctrl + [

---

## **Cell Editor**

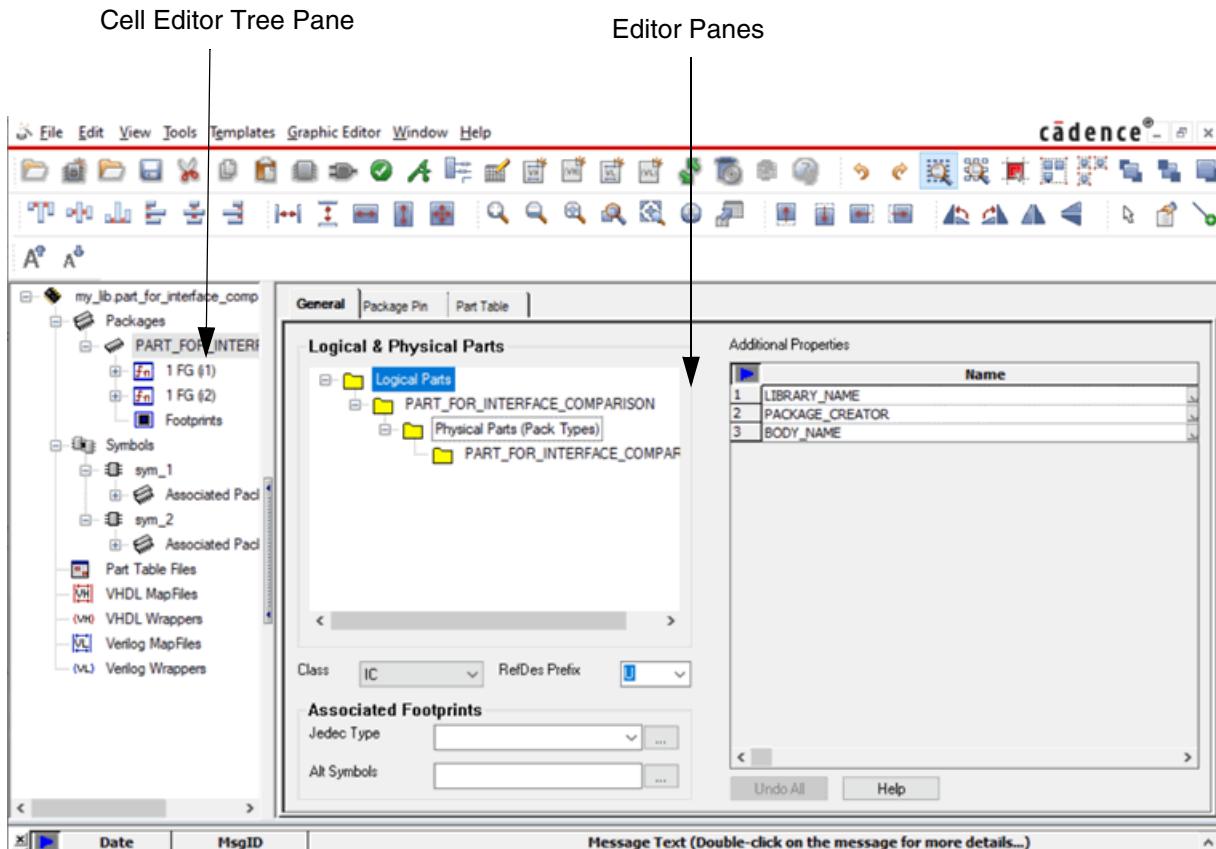
---

Part Developer provides an intuitive and easy way of creating a part and its views. It provides a cell editor through which all the necessary part-creation activities can be accessed. The Cell Editor provides a tree view of the part and the following editors:

- [Package Editor](#)
- [Symbol Editor](#)
- [VHDL Map File Editor](#)
- [VHDL Wrapper File Editor](#)
- [Revision Editor](#)

These editors provide the mechanism to enter and modify the different information about a part. For example, the Package Editor enables you to create and modify packages. These editors are explained in greater detail later in the chapter.

## Cell Editor User Interface



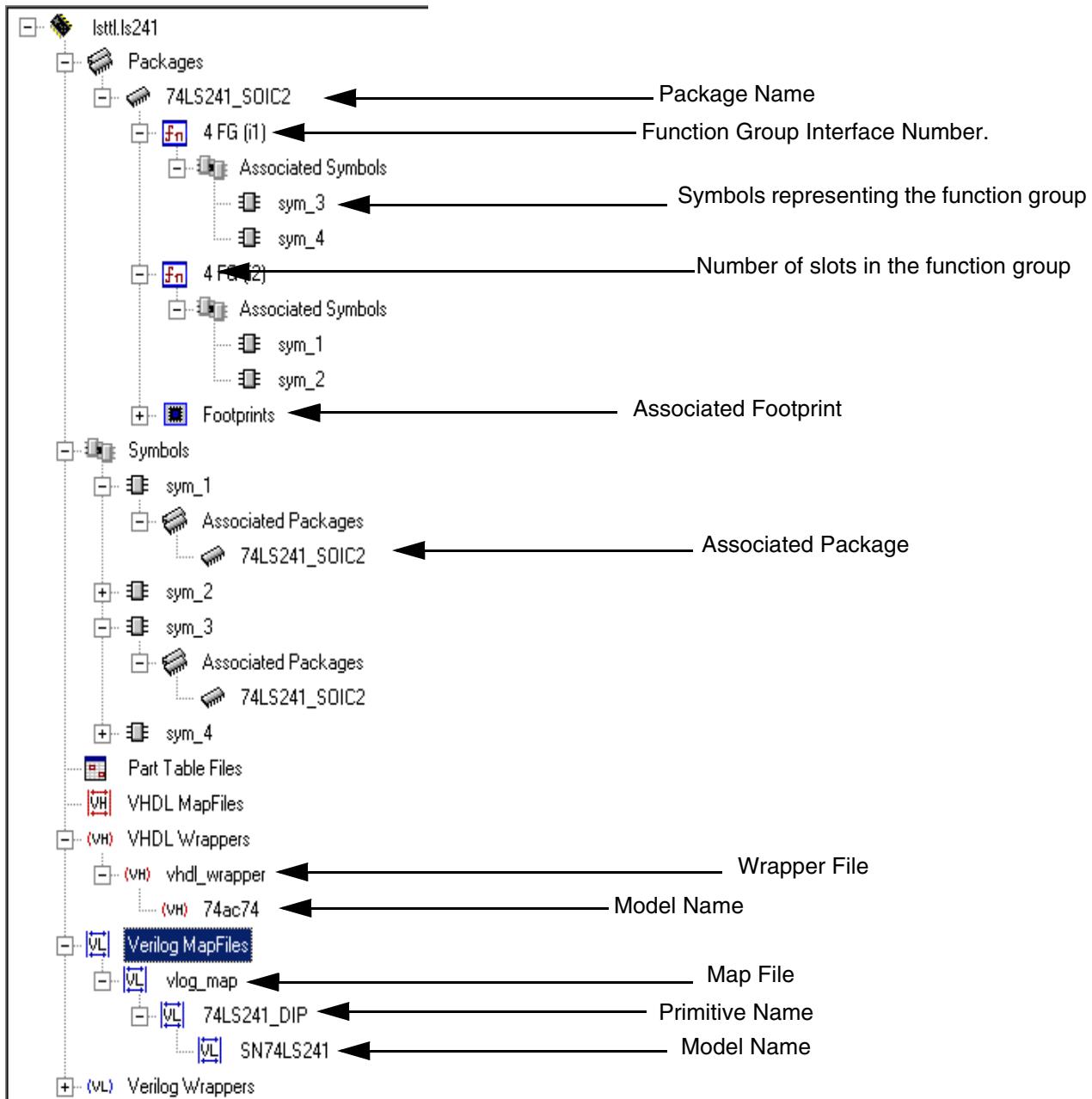
The Cell Editor is divided into two panes:

- The Cell Editor Tree Pane
- The Editor Panes

## Part Developer User Guide

### Cell Editor

#### Cell Editor Tree



The Cell Editor Tree pane shows the part information in a tree structure. The cell name is the top-level entry of the tree. The packages, symbols, the logical pin list, map views, wrapper views, PTF rows, and entity and log files make the first-level nodes of the tree.

Under each package, you can see the different information about a package, such as the number of functions, the number of slots for each function, the symbols that can be packaged into it, and the footprint information. The function/slot information is represented as  $<n>FG[i<n>]$ , where the prefix  $<n>$  represents the number of slots for the given functionality. The suffix  $<n>$  represents the interface number or the function group number.

For example, consider the part LS241. It has two functions, with each function repeated four times. Therefore, in the Cell Editor tree, the following two nodes will appear as:

- $4FG[i1]$
- $4FG[i2]$

Similarly, under each symbol, you can see the packages that are associated with it, i.e. the packages into which the symbol can be packaged.

The *Verilog/VHDL MapFiles* node displays the primitives and the models in each primitive. The *Verilog/VHDL Wrappers* show the wrappers and the models on which the wrappers are based.

The editor pane shows the specific editors. For example, if you click a package name on the Cell Tree pane, the Package Editor will get loaded in the editor pane.

## Viewing Relationships

The Cell Editor Tree enables you to view the relationships that exist between the packages and symbols of the part. The relationship between the packages and symbols of a part is explained from the context of a package.

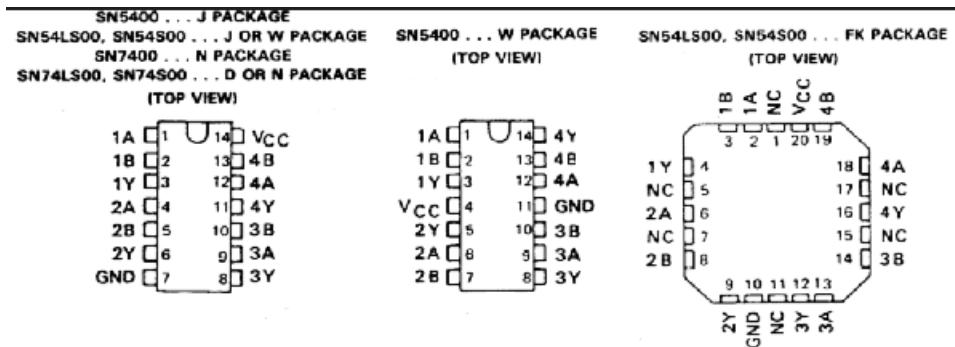
A part can have multiple packages, where typically the packages will differ from each other in terms of the logical-to-physical pin mapping. When symbols are created for a part, the symbols need to be packageable into one or more packages so that they can be used in the front-to-back flow.

For example, consider the part LS00. It comes in a number of packages, such as DIP, SOIC, CFP, and CCC. The following image displays the different packages and their logical-to-physical pin mappings as they appear in the datasheet. Also note that while the J (DIP), W

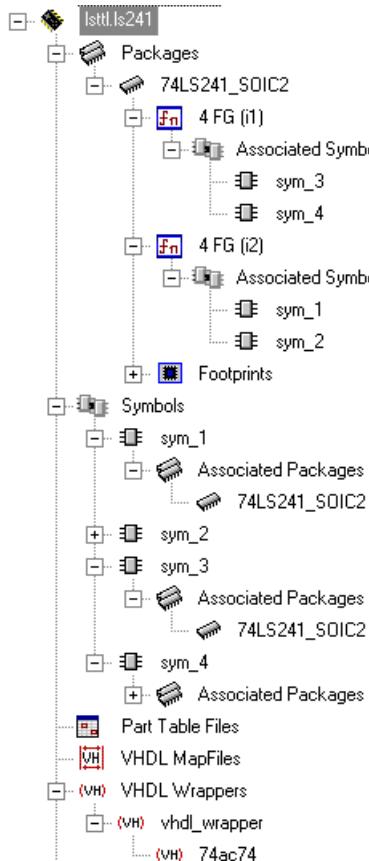
## Part Developer User Guide

### Cell Editor

(CFP), and N (SOIC) packages have 14 pins each, the FK (CCC) package has 20 pins. This is due to the presence of NC pins.



Similarly, other parts can have symbols that are packageable into specific packages. The problem is how do you visually determine which symbol is packageable into what package. The Cell Editor helps you answer that query by showing under a package the symbols that can be successfully packaged into it.



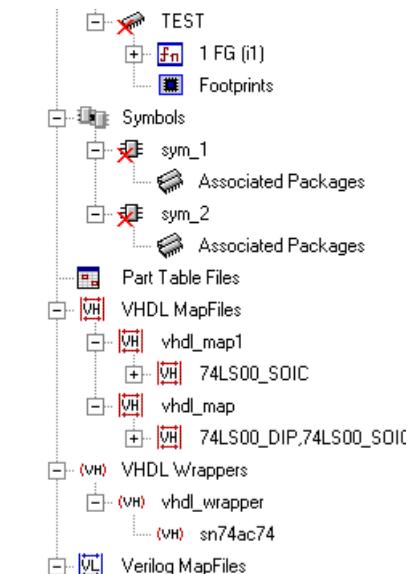
## Part Developer User Guide

### Cell Editor

Another situation is when a part, such as LS241, has multiple functional groups for each package. For such parts, there will be multiple symbols (one symbol each for each functional group) that can be packaged into a package. The Cell Editor helps you in visually identifying such scenarios as well. The tree pane in the Cell Editor will show the functional groups that exist for a package. Under each functional group, you can see the symbols that represent the functional group.

Similarly, the symbol information in the tree view shows which packages are mapped with which symbols.

In case a symbol is not packageable into any of the existing packages or a package exists with no symbol associated with it, those packages and symbols will appear with a red cross on them.



## Package Editor

The Package Editor appears when a package is selected in the Cell Editor tree. The Package Editor provides you the ability to create and modify packages for a part.

The Package Editor has the following pages:

- General
- Package Pin
- Part Table

## General

The General page has the following elements:

### Logical and Physical Parts

The Logical and Physical Parts tree shows the logical and physical parts for a cell.

A logical part defines the logical pins for a part and is mapped to one or more physical parts. A physical part consists of the logical-to-physical pin mapping and set of physical properties. Each primitive entry in the *chips* file represents a physical part. The name of a physical part is either the same as the logical part, or the logical part name suffixed by a package type. The default physical part has the same name as the logical part. The packages that are valid for the specified PART\_NAME appear under the Pack Type entry of the tree.

For example, consider the entry in the `chips.prt` file for the SOIC package of the 7400 part.

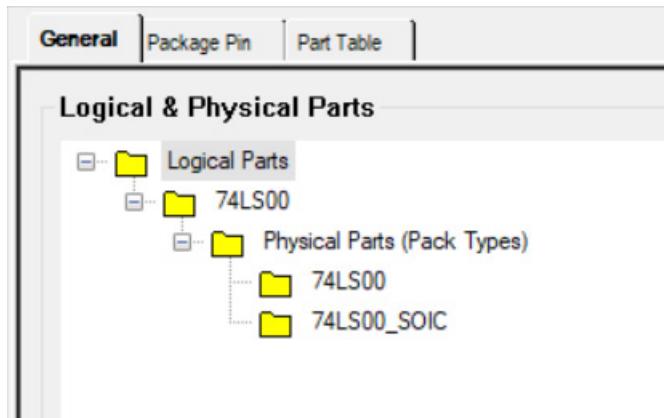
```
FILE_TYPE=LIBRARY_PARTS;
primitive '74LS00','74LS00_SOIC';
pin
  'B'<0>:
    PIN_NUMBER='(13,10,5,2)';
    PIN_GROUP='1';
    INPUT_LOAD='(-0.4,0.02)';
  'A'<0>:
    PIN_NUMBER='(12,9,4,1)';
    PIN_GROUP='1';
    INPUT_LOAD='(-0.4,0.02)';
  '-Y'<0>:
    PIN_NUMBER='(11,8,6,3)';
    OUTPUT_LOAD='(8.0,-0.4)';
end_pin;
body
  PART_NAME='74LS00';
  FAMILY='LSTTL';
  BODY_NAME='LS00';
  DEFAULT_SIGNAL_MODEL='SN74LS00N TI';
```

## Part Developer User Guide

### Cell Editor

```
TECH='74LS';
JEDEC_TYPE='DIP14_3';
PHYS_DES_PREFIX='U';
CLASS='IC';
POWER_PINS='(VCC:14)';
POWER_PINS='(GND:7)';
end_body;
end_primitive;
```

When you load this part in Part Developer, and select the SOIC package in the Cell Editor, the package is displayed in the Package Editor as following:



Note that the value of the PART\_NAME property appears as the child node of the 74LS00 node.

For more information about logical and physical parts, see the chapter *How Packager-XL Selects Names and Parts* in *Packager-XL Reference*.

### Class

Select the type of the part from the list of available part types. The possible types are IC, IO, and Discrete. This information goes into the `chips.prt` file as the CLASS property.

### RefDes Prefix

Select the reference designator prefix for a part from the list of reference designator prefixes. This information is added as a value to the PHYS\_DES\_PREFIX property in the `chips.prt`

file. During packaging, this property is used to write the prefix of the LOCATION property value. For example, if you put the reference designator prefix for a part as U and then use the part three times in your design, the LOCATION values are assigned as U1, U2, and U3. The PHYS\_DES\_PREFIX property is typically used to identify classes of parts such as resistors (R), capacitors (C), and inductors (L).

### Associated Footprints

The Associated Footprints group box has two fields, *Jedec Type* and *Alt Symbols*. These two fields enable you to associate footprint information to the part. You can either manually specify the value of the *Jedec Type* and *Alt Symbols* fields or select the footprint information by browsing. By default, you can select the footprint information from the Cadence-supplied Allegro footprints. After you select the footprints, they will appear under the *Footprints* node in the Cell Editor tree.

**Note:** Part Developer uses the value of the PSMPATH variable for determining the path to the footprint files. This variable is set when you install Part Developer. By default, this variable points to the location where the Cadence-supplied Allegro footprints are stored. You can modify the PSMPATH variable through the Allegro Setup option. For more information, see [Modifying Footprint Information](#) on page 161.

The value of the *Jedec Type* field is added as a value to the JEDEC\_TYPE property in the *chips.prt* file. The value of the *Alt Symbols* field is added as a value to the ALT\_SYMBOLS property in the *chips.prt* file. The JEDEC\_TYPE and ALT\_SYMBOL properties enable Packager XL to select the correct package for the part. You can also extract the physical pin numbers from the footprint information.

### Additional Properties

The Additional Properties grid enables you to add/modify package properties other than CLASS, RefDes, JEDEC\_TYPE, ALT\_SYMBOLS, and POWER\_PINS. The properties for a package are always added to the body section in the *chips.prt* file. For example, consider the body section entry in the *chips.prt* file for the DIP package of the ls00 part. It has the following properties:

```
body
  POWER_PINS='(VCC:14;GND:7)';
  FAMILY='LSTTL';
  PART_NAME='74LS00';
  BODY_NAME='LS00';
  DEFAULT_SIGNAL_MODEL='SN74LS00N TI';
```

## Part Developer User Guide

### Cell Editor

---

```
JEDEC_TYPE='DIP14_3';  
CLASS='IC';  
TECH='74LS';  
end_body;
```

As shown below, all properties with the exception of PART\_NAME, JEDEC\_TYPE, CLASS, and POWER\_PINS appear in the Additional Properties grid. You can add, remove, or modify the properties that get associated with a package through this grid.

Additional Properties		
	Name	Value
1	BODY_NAME	LS00
2	DEFAULT_SIGNAL_MODEL	SN74LS00N T1
3	FAMILY	LSTTL
4	TECH	74LS

### Undo All

Reverses the actions that have been performed since the last save.

### Package Pin

The *Package Pin* page enables you to enter the pin information about a package. Both the physical and the logical pins can be entered through the *Package Pin* page. You can also modify the existing pin information through the *Package Pin* page.

The *Package Pin* page has the following components:

#### Logical Pins

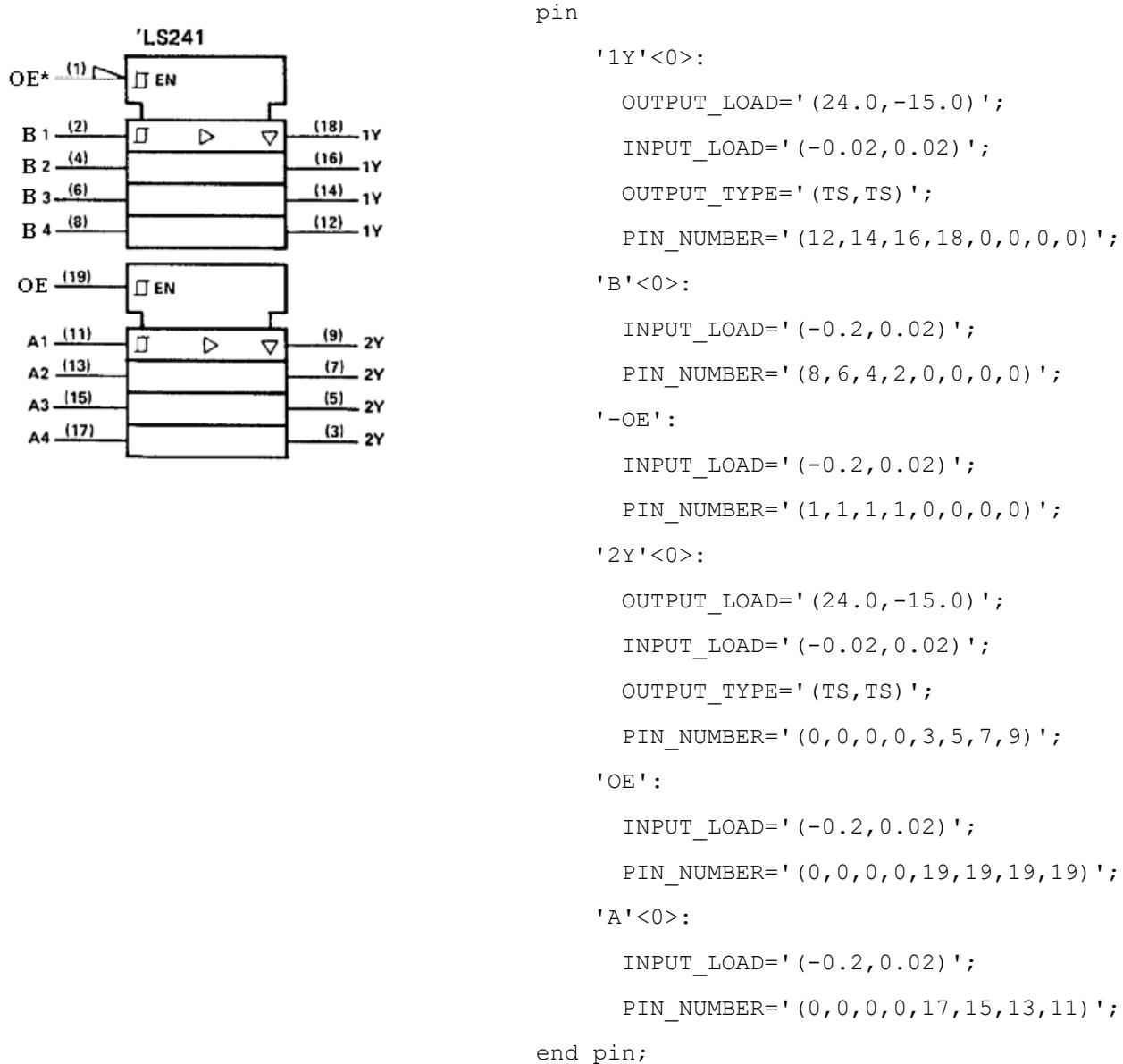
The *Logical Pins* grid shows the logical pin list of a package. The example of the LS241 part is used to explain the *Logical Pins* grid.

The following table shows the graphical representation of LS241 and the pin section of the DIP package of LS241:

## Part Developer User Guide

### Cell Editor

---

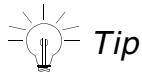


---

As displayed, the pins A, B, and Y are sizeable pins with one enable signal OE and another low-asserted enable signal OE\*. The part has two distinct functions with each function repeated four times. Therefore, as shown in the pin section entry, the part has 8 slots.

## Part Developer User Guide

### Cell Editor



You can determine the number of slots that a part has by seeing the value of the PIN\_NUMBER property of a pin. The value is a comma-separated list. The number of entries in the list represents the number of slots. The number of slots depends on the functions and the number of times they are repeated in a package. Non-zero numbers in the list shows the slots in which the pin is present and the numbers themselves represent the physical pin numbers to which the logical pin is mapped. 0 means that the logical pin does not exist in the slot. For example, LS241 has two unique functions, with each functionality repeated four times in a package. This translates to the package having eight slots.

The Logical Pins grid for LS241 will be as follows:

The screenshot shows the Part Developer Cell Editor interface with the "Package Pin" tab selected. The "Logical Pins" grid on the left lists pins 1 through 6 with their names, types, and assigned slots (S1-S8). The "Physical Pins" grid on the right lists pins 12 through 19 with their mapping to logical pins S1-S8. Buttons for "Map To", "Map", "Unmap", and "Unmap All" are visible between the grids.

	Name	Type	S1	S2	S3	S4	S5	S6	S7	S8	Sized
1	1Y	TS	12	14	16	18					
2	B	INPUT	8	6	4	2					
3	OE*	INPUT	1	1	1	1					
4	2Y	TS				3	5	7	9		
5	OE	INPUT				19	19	19	19		
6	A	INPUT				17	15	13	11		

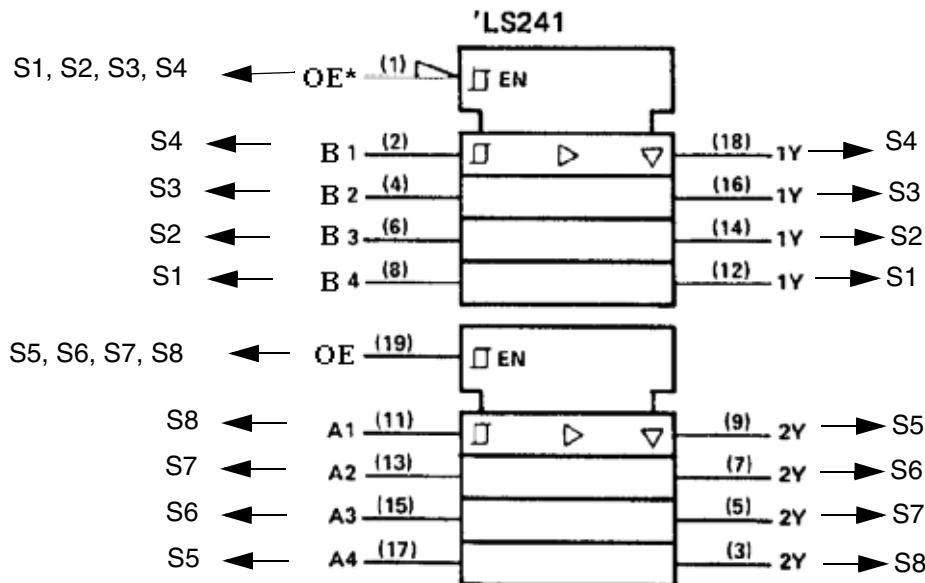
	Number	Mapping	^
		Pin	Function
1	12	1Y	S1
2	14	1Y	S2
3	16	1Y	S3
4	18	1Y	S4
5	8	B	S1
6	6	B	S2
7	4	B	S3
8	2	B	S4
9	1	OE*	S1,S2,S3,S4
10	3	2Y	S5
11	5	2Y	S6
12	7	2Y	S7
13	9	2Y	S8
14	19	OE	S5,S6,S7,S8

Graphically, this is how the Logical Pins grid maps the pin and slot entries for a part:

## Part Developer User Guide

### Cell Editor

---



The *Logical Pins* grid has the following columns:

---

<b>Logical Pins Grid Columns</b>	<b>Description</b>
Name	This shows the logical pin names. For example, the part LS241 has 6 logical pins: A, B, 1Y, 2Y, OE, and OE*. These will appear as separate rows in the <i>Logical Pins</i> grid.
Type	This shows the type of the logical pin. If required, you can modify the pin type for the package through this field.
S1..Sn	This represents the number of slots. A number in a cell under a particular slot number represents the physical pin number that is mapped to the particular slot for the pin. For example, the value 12 under column S1 for pin name 1Y implies that 1Y is present in slot S1 and the physical pin number mapping to this pin and present in S1 is 12.
Sized	This check box determines whether the logical pin is sizeable or not. For LS241, pins A, B, 1Y, and 2Y are sizeable.

## Part Developer User Guide

### Cell Editor

---

Logical Pins Grid Columns	Description
Input Load	This column has two fields, Low and High. The values of Input Low and High fields determine the low and high input load values for the pin in mA. The default values for these fields are taken from Setup.
Output Load	This column has two fields, Low and High. The values of output Low and High fields determine the low and high output load values for the pin in mA. The default values for these fields are taken from Setup.
Check Load	This field determines whether the <code>loading_check</code> Rule Checker rule is executed for the pin. The value for the check is taken from Setup.
Check IO	This field determines whether the <code>inputio_check</code> Rule Checker rule is executed for the pin. The default value is taken from Setup.
Check Dir	This field determines whether direction check will be done for the pin. The default value is taken from Setup.
Check Assert	This field determines whether an assertion check will be done on the pin. The default value is taken from Setup.
Check Output	This field determines whether an output check will be done on the pin. The default value is taken from Setup.
Unknown Loading	The unknown loading property, attached to library components or in your drawings, turns off load checking. When used as a component property, it applies to all pins of the component. As a pin property, it applies to the entire net to which the pin is attached.
Differential Pair	This field displays the Differential Pair.
PIN_DELAY	Specify the value of the <code>PIN_DELAY</code> property for a specific pin type.

## Pins

Pins enables you to do the following to the logical pin list:

- Add
- Global Modify

- Global Rename
- Global Delete
- Set Pin Order
- Retrieve Pin Order

For a step-by-step discussion on how to edit pins, see [Using the Add Pin Dialog Box](#) on page 112 and [Modifying Logical Pins](#) on page 155.

Pins also includes options to set the pin order displayed in the *Logical Pins* grid and retrieve it when required. For the steps to set and retrieve pin order, see [Setting and Retrieving Pin Order](#) on page 121.

## Properties

*Properties* enables you to do the following to package pins:

- Add
- Rename
- Delete

For a step-by-step discussion on how to edit properties, see [Adding Package Pin Properties](#) on page 128, [Adding Symbol Pin Properties](#) on page 136, [Modifying Package Pin Properties](#) on page 167, and [Modifying Symbol Pin Properties](#) on page 172.

## Functions/Slots

*Functions/Slots* enables you to do the following:

- Add slots to a package
- Delete slots to a package
- Modify the distribution of pins across slots

For a step-by-step discussion on how to edit functions, see [Creating Packages](#) on page 121 and [Modifying Pin Lists and Mapping](#) on page 163.

## Generate Symbol(s)

Generates symbols from the given package.

## **Keep Symbols Associated**

Simultaneously updates the pin list of the symbol whenever the package pin list is modified. This ensures that the package-symbol association is never lost.

## **Global Pins**

Displays the list of pins that are applicable to all the slots of a part. POWER, GROUND, and NC are the pin types that constitute global pins. By default, these pins are not placed in the symbols and are placed in the `body` section of the `chips.prt` file.

## **Move**

### ***Global Pins To Logical***

Moves the pins in the *Global Pins* grid to the *Logical Pins* grid. In the `chips.prt` file, the global pins are moved to the `pin` section from the `body` section.

**Note:** When any global pin that is mapped to multiple physical pins, such as a VCC pin mapped to multiple physical pins, is moved, it is converted to the vector bus.

### ***Logical Pins To Global***

Moves the global pins from the *Logical Pins* grid to the *Global Pins* grid. In the `chips.prt` file, the global pins are moved from the `pin` section to the `body` section.

**Note:** In case the global pin is present as a vector pin, right-click on any one of the vector bits and select *Select All Bits*. This selects all the bits of a vector bus. Once all the bits are selected, use the *Move Down* button to move the global pin to the *Global Pins* grid.

## **Global Pin Map**

## ***Edit***

Displays the Edit Global Pins dialog box which shows the mapping between the global pins and the physical pin numbers that are mapped to the global pins and the unmapped physical pin numbers. You can modify the global pin mappings through this dialog box.

### **Remove Errors**

Removes those physical pins from the mapping that are already mapped to other logical/global pins. The physical pin is removed only from global pins that are appearing erroneous. For example, consider two global pins, VCC and GND, mapped to the physical pins 10 and 20, respectively. Now, if you try to map pin 10 to GND, Part Developer will report an error for the GND pin. On selecting Remove Errors, Part Developer will automatically unmap GND from pin number 10. Mapping of pin number 10 with VCC will not be affected.

### **Physical Pins grid**

The *Physical Pins* grid displays the physical pins and its mapping with respect to the logical pins and the slots. You can add new physical pins either manually or by extracting from a footprint. You can extract from a footprint only if the JEDEC\_TYPE is specified.

### **Map**

Maps the logical pins/global pins with physical pins. Select the global pins or the logical pins and the slots for which mapping is to be done. Next, select an equal number of pins from the *Physical Pins* list and click *Map*. The mapping is done as per the order in which the logical and physical pins were selected. For example, if for logical pin A, slot S1, S3, and S2 were selected in that order and the physical pins selected were A3, A2, and A1, then A3 with get mapped to slot S1, A2 with slot S3, and A1 with slot S2.

### **Unmap**

Unmaps the logical to physical pin mapping for a selected logical pin. Similar to mapping, unmapping can be done either on a slot-by-slot basis or for all slots of one or more logical pins.

For example, you can select the cell under slot S1 for pin 1Y and the cell under slot S2 for pin B and click Unmap. This will result in the pins 12 and 6 getting unmapped for logical pins 1Y and B from slots S1 and S2. Alternatively, you can select the logical pin A and click *Unmap*. This will result in unmapping of all the physical pins that are associated with the logical pin A across all the slots.

### **Unmap All**

Unmaps all the global/logical to physical pin mappings.

### **Map To -**

Marks all the slots wherever a logical pin is not present with a -. The - gets translated to 0s in the PIN\_NUMBER property when the part is saved.

### **Filter**

Selects the required physical pins without clicking on them. This is a useful feature when you have a large number of physical pins and you want to select a few of them.

### **Undo All**

Reverses the actions that have been performed since the last save.

### **Footprint**

#### ***Add Physical Pins Manually***

Brings up the Add Physical Pin Numbers dialog box through which you can add physical pin numbers manually. You can add pins in a linear fashion, i.e. a series like 1-3, or in a gridlike manner, such as A1-A10. Grid mode is helpful when you want to manually enter physical pins for large pin-count devices, such as FPGAs.

#### ***Extract from Footprint***

Extracts physical pin numbers from the specified footprint.

**Note:** The footprints are extracted from the \*.psm files.

#### ***Select Using Ptf & Extract From Footprint***

Reads the ptf file for the JEDEC\_TYPE value and extracts the physical pins from the footprint.

#### ***Select and Extract from Footprint***

Brings up the *Browse Jedec Type* dialog box. You need to select the footprint from which the physical pin numbers will be extracted.

### **Verify with all Footprints**

Verifies the physical pin list with the pin list of the footprints specified as JEDEC\_TYPE and Alt\_Symbols property values.

### **Select Using Ptf & Verify With Footprint**

Verifies the physical pin list with the pin list of the footprint mentioned as the JEDEC\_TYPE value in the part table file.

### **Select and Verify with Footprints**

Verifies the physical pin list with the pin list of the selected footprint.

## **Part Table**

The *Library Level PTF* displays the library-level part table file details, such as the physical location of the file and the key and injected properties and their values. It has the following components:

### **Global Properties**

This grid displays the global properties present in the part table file.

### **Part Rows**

This grid displays the entries in the part table file that are relevant to the selected part and its packages.

## **Symbol Editor**

The Symbol Editor helps you view the complete information about a symbol. The symbol graphic is displayed on the Symbol Editor canvas. You can manipulate the symbol in the graphical mode by manipulating the symbol in a variety of ways. The options to manipulate the symbols are available through the *Graphic Editor* menu option. The same options are also available through the toolbars as well.

The Symbol Editor has the following elements:

- [General](#) page
- [Symbol Pins](#) page
- [Find](#) page
- [Symbol Editor Canvas](#)

## General

The *General* page describes the symbol properties and symbol text and their attributes. It has the following components:

### Properties

The Properties grid displays the symbol properties. It has the following columns:

Properties Grid Columns	Description
Name	Displays the names of the symbol properties. By default, one or more of the following properties can be selected from the list: <ul style="list-style-type: none"><li>■ PACK_TYPE</li><li>■ VALID_PACK_TYPE</li><li>■ JEDEC_TYPE</li><li>■ ALT_SYMBOLS</li><li>■ PART_NAME</li><li>■ \$LOCATION</li><li>■ SPLIT_INST</li><li>■ SPLIT_INST_NAME</li></ul> To know more about these properties, see <i>PCB Systems Properties Reference</i> .
Value	Displays the values of the symbol properties.

## Part Developer User Guide

### Cell Editor

---

<b>Properties Grid Columns</b>	<b>Description</b>
Visibility	Determines the display characteristics of the property and its value. The possible values are Invisible, Name, Value, and Both. If the <i>Visibility</i> is set to invisible, then both the property name and its value are hidden. Setting <i>Visibility</i> to Name results in only the property name being visible on the symbol. If the <i>Visibility</i> is set to Value, only the property value is visible on the symbol. Setting the <i>Visibility</i> to Both results in both property name and its value being visible on the symbol.
Location	Determines where the symbol properties will be displayed.
Text Height	Determines the height of the property name and its value. The default value for text height is taken from Setup.
Alignment	Determines the alignment of the property. The possible values are left, right, and center.
Rotation	Determines the angle at which the property is displayed on the symbol. The possible values are 0, 90, 180, and 270.
Parameter	Determines whether the property is written as a parameter when the Verilog netlist is generated for the part. This is generated when the part is saved and the entity view is written.
Color	Determines the color in which the property is displayed. The possible values are Mono, Red, Green, Blue, Yellow, Orange, Salmon, Violet, Brown, Skyblue, White, Peach, Pink, Purple, Aqua, and Gray.
X and Y	Determines the position of the property from the origin.

### Text

The *Text* grid shows all the text associated with the symbol. It has the following columns:

<b>Text Grid Columns</b>	<b>Description</b>
Text	Displays the text on the symbol.

## Part Developer User Guide

### Cell Editor

---

<b>Text Grid Columns</b>	<b>Description</b>
Location	Determines where the symbol text is displayed. The figure displayed below describes the possible locations:
Text Height	Determines the height of the text on the symbol.
Alignment	Determines the alignment of the property. The possible values are left, right, and center.
Rotation	Determines the angle at which the property is displayed on the symbol. The possible values are 0, 90, 180, and 270.
Color	Determines the color in which the property is displayed. The possible values are:
X and Y	Determines the position of the property from the origin.

### **Undo All**

Reverses the actions that have been performed since the last save.

## **Symbol Pins**

The *Symbol Pins* page enables you to enter the symbol pins and determine the symbol size. You can also modify the existing pin information and symbol size through this page.

It has the following components:

### **Logical Pins**

The *Logical Pins* grid shows all the symbol pins and properties associated with the pins. By default, it has the following columns:

<b>Logical Pins Grid Columns</b>	<b>Description</b>
Name	Displays the name of the symbol pins.

## Part Developer User Guide

### Cell Editor

---

#### Logical Pins Grid Columns

#### Description

Text	Displays the value of the PIN_TEXT property. If the <i>Use Pin Name As Pin Text</i> option is enabled in Setup options at part creation, the pin names will appear as pin text.
Pin Type	Displays the type of the pin, such as Input, Output and so on. If there is an associated package with the symbol, the pin type information is read from the package pin list. This column is non-editable.
Sized	Displays whether the pin is sized or not. This field is non-editable.
Location	Determines the location of the symbol pin. The possible values are left, right, top, and bottom.
Position	Determines the position of the pin with respect to the origin. For example, a pin with location value as left and position value as 7 will mean that the pin is placed on the seventh grid at the left boundary above the origin.
Pin Delay	Specify the value of the PIN_DELAY property for a specific pin type.

#### Preserve Pin Position

Determines whether the pin positions and their associated properties and pin texts will be preserved when the symbol outline is modified. If the option is not selected, the pin positions, their properties, and pin texts are adjusted dynamically when symbol outline changes. Otherwise, the pin positions, their properties, and pin texts do not change with the change in the symbol outline.

Whenever a new symbol is created, this option is unchecked by default. This allows Part Developer to dynamically shift the pins on the symbol outline as and when new pins are added.

**Note:** Part Developer will extend the symbol outline when new pins are added. However, no symbol outline contraction is done when symbol pins are deleted. You need to manually resize the symbol outline.

When an existing symbol is loaded, this option is selected by default. This is done to ensure that graphics associated with a symbol pin are not destroyed by automatic movement of pins.

This also ensures that in case of horizontal/vertical extension of symbol outline due to addition of pins, the pins on top/bottom and/or left/right of the symbol remain bounded to the symbol outline.

### **Symbol Outline**

Determines the length and breadth of the symbol with respect to the origin. You can control the dimensions of the symbol outline through the Left, Right, Top, and Bottom fields.



***Changing the symbol outline when the Preserve Pin Position is selected may result in pins losing their connection with the symbol outline. In such cases, you need to manually reconnect the pins to the symbol outline by shifting the pin positions.***

### **Move Pins**

Enables you to move pins selected in the *Logical Pins* grid to left, right, up, or down directions by one or more grids. The arrow keys move the pin in the specified direction by one grid. In case you want to move pins by more than one grid, use the *Move* button in the middle of the four arrow keys.

For more information, see [Move Pin](#) on page 423.

### **Set Origin**

Enables you to set the origin. To set the origin, click *Set Origin* and then click on the desired location on the Symbol Editor canvas. The origin will be shifted to the clicked location and the symbol will be re-drawn around it.

### **Set Size**

Enables you to set the value of the SIZE property for sizeable parts. For more information, see [Creating Sizeable and HAS\\_FIXED\\_SIZE Symbols](#) on page 138.

### **Pins**

Enables you to add, rename, modify, and delete the pins to the part pin list. Additionally, you can also modify the attributes of the symbol pins and the attributes of the PIN\_TEXT property.

## Part Developer User Guide

### Cell Editor

---

The following table details where you can get more information about the functionality provided by the *Pins* button.

<b>Pin Functions</b>	<b>For More Information, see...</b>
Add	<a href="#">Add Pin</a> on page 410
Global Rename	<a href="#">Rename Pin</a> on page 416
Global Modify	
Global Delete	<a href="#">Delete Symbol Pin</a> on page 419
Attributes	<a href="#">Symbol Pin Attributes</a> on page 419
Pin Text Attributes	<a href="#">Symbol Pin Property Attributes</a> on page 441
Associate/ Unassociate Pin Text	<a href="#">Map Pin Name and Text</a> on page 421
Adjust symbol outline to Pin Text	

## Properties

Enables you to add, rename, or delete properties associated with a symbol pin. The following table details where you can get more information about the functionality provided by the *Properties* button:

<b>Properties Functions</b>	<b>For More Information, see...</b>
Add	<a href="#">Add Properties</a> on page 417
Rename	<a href="#">Rename Symbol Pin Property</a> on page 422
Delete	<a href="#">Delete Symbol Pin Property</a> on page 422
Attributes	<a href="#">Symbol Pin Attributes</a> on page 419

## Undo All

Reverses the actions that have been performed since the last save.

## Find

The *Find* page displays options for specifying a filter to quickly locate objects on the Symbol Editor canvas.

The Find page has the following elements:

- *Filter for Object Selection* group box
- *Find By Name* group box

### ***Filter for Object Selection***

The *Filter for Object Selection* group box enables you to specify filter criteria for selecting objects on the Symbol Editor canvas. When the filter criteria are set and you click the *Set Filter* button and make a selection on the Symbol Editor canvas, all the objects that meet the criteria are selected.

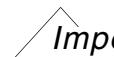
You can select one or more options from the following list to specify the filter criteria:

- Line
- Arc
- Rectangle
- Polygon
- Pin Name
- Pin Shape
- Pin Property
- Text Shape
- Text
- Circle
- Polyline
- Image
- Group
- Custom Shape
- Symbol Property

The *All On* and *All Off* buttons enable you to select or clear all the options in a single step.

 **Important**

In Part Developer, all graphical objects except shapes are stored as lines and arcs. Therefore, the *Rectangle*, *Polygon*, and *Polyline* options in the Filter for Object Selection group box work only if you have not reloaded a part.

 **Important**

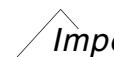
Any object that is part of a group cannot be searched independently. To find such an object, you need to select the *Group* check box in addition to the object type.

### **Find By Name**

In the *Find By Name* group box, you can specify a search criterion based on attribute names and their values. Clicking *Find* highlights the symbol objects that meet the criterion.

The following attributes are supported:

- Text
- Pin name
- Pin shape
- Custom shape
- Property
- Text shape

 **Important**

Find Filter does not distinguish between any text added using the *Text* option on the Draw menu or the *Add Text* tool button and any text that is part of a text shape. Therefore, if a symbol contains the same text, for example, *cadence*, both as a text entry and as part of a text shape, selecting the *Text* option in the Find By Name group box and searching for *cadence* finds both the occurrences of *cadence*.

### **Symbol Editor Canvas**

The Symbol Editor canvas displays the symbol selected in the tree pane. All changes that you make to the symbol through the Symbol Editor pages are immediately reflected on the

Symbol Editor canvas. In addition, you can use the *Graphic Editor* menu options to modify the displayed symbol in a variety of ways.

To view the information on default Symbol Editor, [\*Symbol Editor User Guide\*](#).

## **Symbol Editor Grid Settings and Behavior**

The Symbol Editor canvas supports the following grids:

- Pin
- Non-pin

### **Pin Grid**

The pin grid is the default grid and coarser than the non-pin grid. In the default grid setup, all objects—both pins and non-pin objects—are placed on the coarse grid.

**Note:** When opening a symbol, Part Developer displays an error message if any symbol pin is off grid with respect to the pin grid.

### **Non-Pin Grid**

The non-pin grid is finer than the pin grid.

For information on changing the density of the finer grid, see [Non-pin grid factor](#) on page 93.

You can use the following tool buttons to hide grid lines, display the pin grid, or display the finer grid:



Hides grid lines or  
displays the coarse  
grid



Displays the finer  
grid

### Moving Objects on the Symbol Editor Canvas Using Arrow Keys

You can use the arrow keys on your keyboard to move objects on the editing canvas. Moving of pins and non-pin objects on the Symbol Editor canvas by using arrow keys depends on the visible grid and the selection of *Snap to Grid* as tabulated below:

<b>Grid and Snap to Grid Selection</b>	<b>Pins</b>	<b>Non-Pin Objects</b>	<b>Pins and Non-Pin Objects</b>
Pin grid visible and <i>Snap to Grid</i> selected	Move on the pin grid	Move on the pin grid	Move on the pin grid
Non-pin grid visible and <i>Snap to Grid</i> selected	Move on the pin grid	Move on the non-pin grid	Move on the pin grid
Pin grid visible and <i>Snap to Grid</i> deselected	Move on the pin grid	Nudge	Pins move on the pin grid.  Non-pin objects move in pin-grid units, maintaining the distance from the pin.
Non-pin grid visible and <i>Snap to Grid</i> deselected	Move on the pin grid	Nudge	Pins move on the pin grid.  Non-pin objects move in pin-grid units, maintaining the distance from the pin.

## VHDL Map File Editor

The VHDL map file editor enables you to create/modify VHDL map files. The map file editor appears in the right pane of the cell tree when you create a new map file or load an existing map file for modification.

The VHDL Map file editor has two property sheets:

- General
- Mapping

## General

The *General* property sheet appears by default when the VHDL map file editor is loaded in the cell tree. It contains the following elements:

### Model Name

Displays the name of the VHDL model. This field is non-editable.

### Model Alias

Specify the model names that share the same model ports with the same mode/type, port order, and generics.

### Port Order

Displays the exact order of ports as they appear in the entity. This is a non-editable field.

### Select Generic

Displays the VHDL model's generics, their types and values, and whether the generics are annotated on the symbols.

### Annotate Generics

Launches the [Annotate Generics](#) dialog box that enables you to annotate the generics to the symbols. You can assign new values to the generics before annotating them to the symbols. See [Annotate Generics](#) on page 438 for more details.

### Binding Information

Determines whether to bind the VHDL model and the symbol. That is, determine whether to bind the symbol with one specific architecture (behavior). If you decide to bind the VHDL model and the symbol, then the binding statement goes into the wrapper. This is an optional step.

**Note:** Since each view of the model is essentially a specific architecture, selecting the VHDL model by using the lib:cell:view method automatically fills in the binding statement. If you specify the actual physical path to the VHDL model, then you have to explicitly enter the binding information. If you decide not to enter the binding information during wrapper

creation, then the binding information has to be specified later in the simulation flow. However, not providing the binding information in the wrapper provides the freedom to use the same map file for different architectures.

## **Libraries and Packages**

Displays the VHDL libraries that are included for the VHDL model file. Both of these list boxes are non-editable.

## **Mapping**

The *Mapping* page enables you to map the model ports to logical pins. It has the following elements:

### **Automatically Update Pin Mode**

Maps pins to ports of dissimilar mode types. For example, you can map a logical pin, say A, of mode IN with a model port, say AA, of type OUT. After the mapping is done, the mode of the logical pin is updated with the model port mode. Therefore, after the mapping, the mode of the logical pin will change from IN to OUT. Part Developer will not let you map pins of dissimilar mode types if this option is not checked. Part Developer puts the VHDL\_MODE property on all the symbol pins whose modes have changed due to automatic update.

### **Automatically Update Pin Type**

Enables you to map pins to ports of dissimilar types. For example, by checking this option, you can map a logical pin, say A, of type STD\_LOGIC with a model port, say AA, of type OUT. After the mapping is done, the type of the logical pin is updated with the model port type. Therefore, after the mapping the type of the logical pin will change from STD\_LOGIC to OUT. Part Developer will not let you map pins of dissimilar mode types if this option is not checked. Depending on whether the pins are scalar or vector, Part Developer puts the VHDL\_SCALAR\_TYPE or VHDL\_VECTOR\_TYPE property on all the symbol pins whose types have changed due to automatic update.

## **Logical Pin List**

Displays the logical pins present in the package, their modes, types, and the slots in which they are present. Pin mode is determined by the value of the VHDL\_MODE property on the symbol pins. If the property is not found, then the `chips.prt` is read to determine the pin

mode. The VHDL\_SCALAR\_TYPE and VHDL\_VECTOR\_TYPE properties are read to determine the pin type for scalar and vector pins, respectively.

### **Model Port List**

Displays the ports in the VHDL model, their modes, types, the logical pin and the slot in which they are mapped.

### **Map**

Maps logical pins with model ports. In the *Logical Pins* list, select the logical pins and the slots for which the model port is to be mapped. Next, from the *Model Port List*, select the ports and click *Map*. This maps the model ports to the logical pins for the selected slot. The mapping of the model ports with the logical pins is done as per the order in which they were selected. For example, if for logical pin A, slots S1, S3, and S2 were selected in that order and the model ports selected were A3, A2, and A1, then A3 will get mapped to slot S1, A2 with slot S3, and A1 with slot S2.

### **Auto Map**

Automatically maps the logical pins to model ports. If there are some existing pins that are already mapped, then you will be asked if the currently mapped pins should be preserved or not. Automapping maps the model port to the first slot of the corresponding logical pin. Manual mapping needs to be done to map the model ports to other slots.

Automapping is done as per the following rules:

- If pin name is the same as model port name.
- If the pin and the model port names are same, but case is different.
- If the pin and the model port names are same, but assertion is different. For example, automapping will be done in the following cases:

#### **Pin List**

DS\*

DS\*

#### **Model Port List**

DS

ds\_n

- If the logical pin has / in the pin name and the model port has \_. For example, automapping will be done in the following cases:

<b>Pin List</b>	<b>Model Port List</b>
R/W	r_w
R/W*	r_w_n

- If the logical pin and the model port names match at the beginning of their names. For example, automapping will happen for the following:

<b>Pin List</b>	<b>Model Port List</b>
WR	wr1
CS1*	cs_n

### **Unmap**

Removes the mapping between model ports and logical pins for pins selected in the *Logical Pins* list.

### **Unmap All**

Removes the pin-to-port mapping for all the pins.

## **VHDL Wrapper File Editor**

The VHDL wrapper file editor enables you to create and modify VHDL map files. The wrapper file editor appears in the right pane of the cell tree when you create a new wrapper or load an existing wrapper for modification.

The VHDL wrapper file editor has two pages:

- General
- Mapping

## General

The General page appears by default when the VHDL wrapper file editor is loaded in the cell tree. It contains the following elements:

### Model Name

Displays the name of the VHDL model. This field is non-editable.

### Select Generic

Displays the VHDL model's generics, their types and values, and whether the generics are annotated on the symbols.

### Annotate Generics

Launches the [Annotate Generics](#) dialog box that enables you to annotate the generics to the symbols. You can assign new values to the generics before annotating them to the symbols. See [Annotate Generics](#) on page 438 for more details.

### Binding Information

Determines whether to bind the VHDL model and the symbol. That is, determine whether to bind the symbol with one specific architecture (behavior). If you decide to bind the VHDL model and the symbol, then the binding statement goes into the wrapper. This is an optional step.

**Note:** Since each view of the model is essentially a specific architecture, selecting the VHDL model by using the lib:cell:view method automatically fills in the binding statement. If you specify the actual physical path to the VHDL model, then you have to explicitly enter the binding information. If you decide not to enter the binding information during wrapper creation, then the binding information has to be specified later in the simulation flow. However, not providing the binding information in the wrapper provides the freedom to use the same map file for different architectures.

### Libraries and Packages

Displays the libraries that are included for the VHDL model file. Both of these list boxes are non-editable.

## Mapping

The Mapping page enables you to map the model ports to logical pins. It has the following elements:

### Automatically Update Pin Mode

Maps pins to ports of dissimilar mode types. For example, you can map a logical pin, say A, of mode IN with a model port, say AA, of type OUT. After the mapping is done, the mode of the logical pin is updated with the model port mode. Therefore, after the mapping, the mode of the logical pin will change from IN to OUT. Part Developer will not let you map pins of dissimilar mode types if this option is not checked. Part Developer puts the VHDL\_MODE property on all the symbol pins whose modes have changed due to automatic update.

### Automatically Update Pin Type

Maps pins to ports of dissimilar types. For example, by checking this option, you can map a logical pin, say A, of type STD\_LOGIC with a model port, say AA, of type OUT. After the mapping is done, the type of the logical pin is updated with the model port type. Therefore, after the mapping, the type of the logical pin will change from STD\_LOGIC to OUT. Part Developer will not let you map pins of dissimilar mode types if this option is not checked. Depending on whether the pins are scalar or vector, Part Developer puts the VHDL\_SCALAR\_TYPE or VHDL\_VECTOR\_TYPE property on all the symbol pins whose types have changed due to automatic update.

### Logical Pin List

Displays the ports, their modes, and types. Pin mode is determined by the value of the VHDL\_MODE property on the symbol pins. If the property is not found, then the `chips.prt` is read to determine the pin mode. The `VHDL_SCALAR_TYPE` and `VHDL_VECTOR_TYPE` properties are read to determine the pin type for scalar and vector pins, respectively.

### Model Port List

Displays the ports in the VHDL model, their modes, types, and the logical pin to which they are mapped.

## Map

Maps logical pins with model ports. In the *Logical Pins* list, select the logical pins to which the model ports are to be mapped. Next, from the *Model Port List*, select the ports and click **Map**. This maps the model ports to the logical pins. The mapping of the model ports with the logical pins is done as per the order in which they were selected.

### Auto Map

Automatically maps the logical pins to model ports. See [Auto Map](#) on page 70 for details.

### Unmap

Removes the mapping between model ports and logical pins for pins selected in the *Logical Pins* list.

### Unmap All

Removes the pin-to-port mapping for all the pins.

## Verilog Map File Editor

The Verilog Map file editor enables you to create/modify Verilog map files. The map file editor appears in the right pane of the cell tree when you create a new map file or load an existing map file for modification.

The Verilog Map file editor has two pages:

- General
- Mapping

### General

The *General* page appears by default when the Verilog Map file editor is loaded in the cell tree. It contains the following elements:

#### Model Name

Displays the name of the Verilog model. This field is non-editable.

## **Model Alias**

Specify the model names that share the same model ports with the same mode, port order, and parameters.

## **Port Order**

Displays the exact order of ports as they appear in the module. This is a non-editable field.

## **Select Parameters**

Displays the Verilog model's parameters, their values, and whether the parameters are annotated on the symbols.

## **Annotate Parameters**

Launches the [Annotate Parameters](#) dialog box that enables you to annotate the generics to the symbols. You can assign new values to the generics before annotating them to the symbols. See [Annotate Parameters](#) on page 439 for more details.

## **Mapping**

The Mapping page enables you to map the model ports to logical pins. It has the following elements:

### **Automatically Update Pin Mode**

Enables you to map pins to ports of dissimilar mode types. For example, you can map a logical pin, say A, of mode INPUT with a model port, say AA, of type OUTPUT. After the mapping is done, the mode of the logical pin is updated with the model port mode. Therefore, after the mapping, the mode of the logical pin will change from INPUT to OUTPUT. Part Developer will not let you map pins of dissimilar mode types if this option is not checked. Part Developer puts the VLOG\_MODE property on all the symbol pins whose modes have changed due to automatic update.

### **Logical Pin List**

Displays the logical pins present in the selected packages, their modes, and the slots in which they are present. Pin mode is determined by the value of the VLOG\_MODE property on the

symbol pins. If the property is not found, then the `chips.prt` is read to determine the pin mode.

### **Model Port List**

Displays the ports in the Verilog model, their modes, types, the logical pin, and the slot in which they are present.

### **Map**

Maps logical pins with model ports. In the *Logical Pins* list, select the logical pins and the slots for which the model port is to be mapped. Next, from the *Model Port List*, select the ports and click *Map*. This maps the model ports to the logical pins for the selected slot. The mapping of the model ports with the logical pins is done as per the order in which they were selected. For example, if for logical pin A, slots S1, S3, and S2 were selected in that order and the model ports selected were A3, A2, and A1, then A3 will get mapped to slot S1, A2 with slot S3, and A1 with slot S2.

### **Auto Map**

Automatically maps the logical pins to model ports. See [Auto Map](#) on page 70 for details.

### **Unmap**

Removes the mapping between model ports and logical pins for pins selected in the *Logical Pins* list.

### **Unmap All**

Removes the pin-to-port mapping for all the pins.

## **Verilog Wrapper File Editor**

The Verilog Wrapper file editor enables you to create/modify Verilog map files. The map file editor gets loaded in the right pane of the cell tree when you create a new map file or load an existing map file for modification.

The Verilog Wrapper file editor has two pages:

- General
- Mapping

## General

The *General* page appears by default when the Verilog Wrapper file editor is loaded in the cell tree. It contains the following elements:

### Model Name

Displays the name of the Verilog model. This field is non-editable.

### Select Parameters

Displays the Verilog model's parameters, their values, and whether the parameters are annotated on the symbols.

### Annotate Parameters

Launches the [Annotate Parameters](#) dialog box that enables you to annotate the parameters to the symbols. You can assign new values to the parameters before annotating them to the symbols. See [Annotate Parameters](#) on page 439 for more details.

## Mapping

The Mapping page enables you to map the model ports to logical pins. It has the following elements:

### Mapping by Position

This option is enabled only when an existing part which a wrapper has model ports to logical pins mapping done by position is loaded. You can modify the wrappers as required. However, you cannot create wrappers where the mapping is done by position.

### Automatically Update Pin Mode

Maps pins to ports of dissimilar mode types. For example, you can map a logical pin, say A, of mode INPUT with a model port, say AA, of type OUTPUT. After the mapping is done, the

mode of the logical pin is updated with the model port mode. Therefore, after the mapping, the mode of the logical pin will change from INPUT to OUTPUT. Part Developer will not let you map pins of dissimilar mode types if this option is not checked. Part Developer puts the VLOG\_MODE property on all the symbol pins whose modes have changed due to automatic update.

### **Logical Pin List**

Displays the logical pins and their modes. Pin mode is determined by the value of the VLOG\_MODE property on the symbol pins. If the property is not found, then the `chips.prt` is read to determine the pin mode.

### **Model Port List**

Displays the ports in the Verilog model, their modes, and the logical pin to which they are mapped.

### **Map**

Maps logical pins with model ports. In the *Logical Pins* list, select the logical pins to which the model ports are to be mapped. Next, from the *Model Port List*, select the ports and click *Map*. This maps the model ports to the logical pins. The mapping of the model ports with the logical pins is done as per the order in which they were selected.

### **Auto Map**

Automatically maps the logical pins to model ports. See [Auto Map](#) on page 70 for details.

### **Unmap**

Removes the mapping between model ports and logical pins for pins selected in the *Logical Pins* list.

### **Unmap All**

Removes the pin-to-port mapping for all the pins.

## **Row/Column Shown/Hidden Indicator**

The first header of a grid provides a visual indication of whether all the rows/columns are displayed in the grid. If all the rows/columns are visible, then the filter viewer appears in blue. In case some rows/columns are hidden, then the viewer appears in green.

**When All Rows/  
Columns Are Visible**    **When Some Columns/Rows  
Are Hidden**



## **Part Developer User Guide**

### Cell Editor

---

---

# Configuring Part Developer

---

You can configure Part Developer through the following options in the *Tools* menu:

- Setup
- Configuration

## Setup

The *Setup* option enables you to:

- Specify values for fields of the Part Developer user interface. These values are used by Part Developer as default values while creating the different views. For example, you can specify values for symbol texts such that whenever a text is added to a symbol, it appears on the symbol at 180 degrees to the symbol body, is white in color, and 0.05 inches in height.

**Note:** The default values can be modified whenever required.

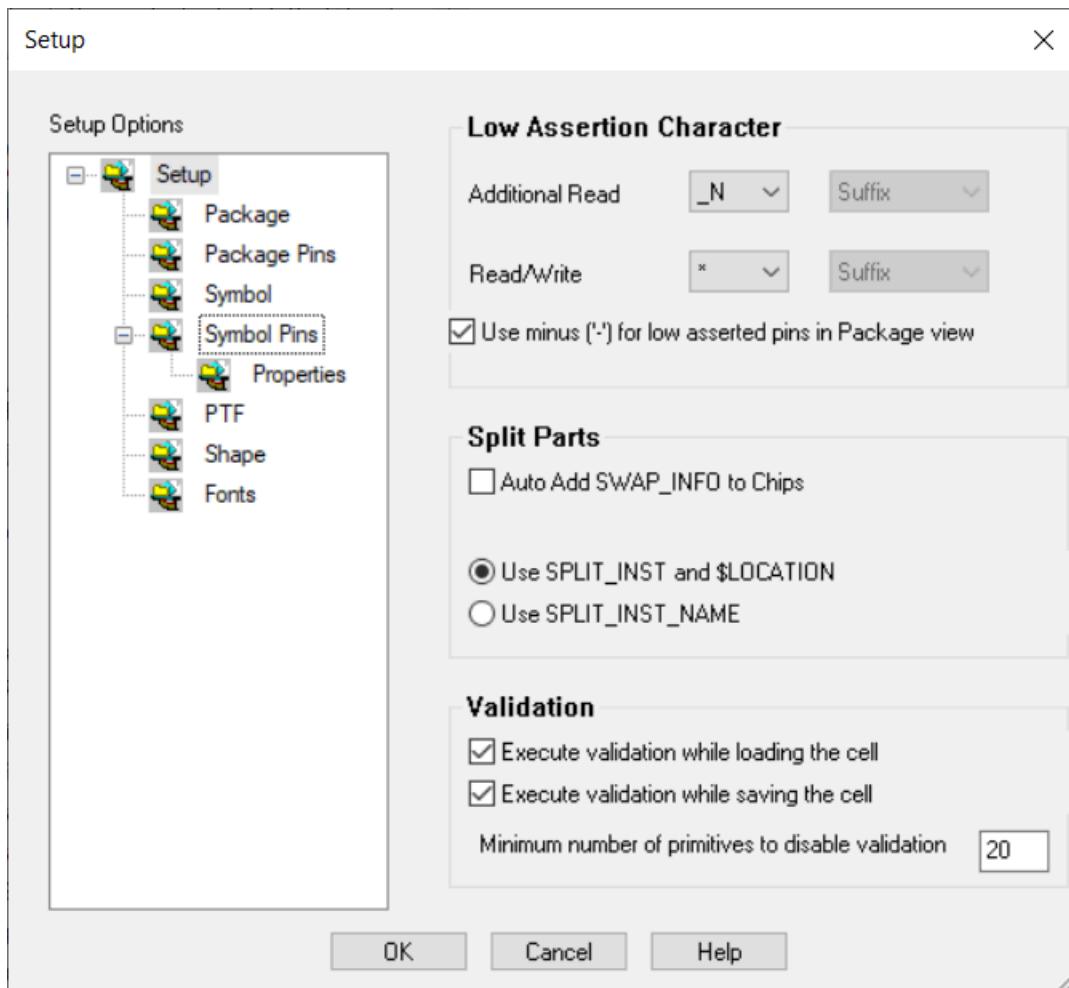
- Configure which pin-name suffix to use for making a pin as low-asserted when reading and writing a part.
- Determine which properties to put for split parts.
- Specify the default properties to be added to symbols, packages, symbol pins, and package pins.
- Specify the attributes of different properties.
- Determine whether to add vector pins in expanded or collapsed form.
- Determine the position and load values of types of pins on symbols and packages, respectively.

To set up Part Developer:

1. Load a project in Part Developer.

**2. Choose *Tools – Setup*.**

The Setup Options dialog box displays.



The *Setup Options* UI is divided into two panes. The left pane displays the different views in a tree view while the right pane displays the options that you can set for a particular view.

## Understanding the Setup Options Tree

The *Setup Options* tree has *Setup* as the root node and *Package*, *Package Pins*, *Symbol*, *Symbol Pins*, *PTF*, and *Shape* as the leaf nodes.

The *Setup* node provides the following:

- Assigning a low-assertion character

- Assigning default properties for split parts
- Determining whether validation needs to be run for a part

The *Package*, *Package Pins*, *Symbol*, *Symbol Pins*, and *Shape* nodes allow you to specify values that are used by Part Developer while creating packages, symbols, and shapes. The *PTF* node enables you to specify the default values for the part table files for the cells.

## Setting Up Defaults for Low-Asserted Pins and Split Parts

To set up Part Developer for handling low-asserted pins and split parts, select the *Setup* node.

The right pane displays the two group boxes, one for handling the low-asserted pins and the other for handling split parts.

### Low-Assertion Character

The setting in this grid box determines the pins that Part Developer will treat as low-asserted while reading and writing a part. Part Developer treats a pin as low-asserted either on the basis of the pin name suffix or the ‘-’ prefix in the pin name.

#### *Read/Write*

Determines whether the *\_N* or *\** suffix in pin names should be used to treat pins as low-asserted when reading or saving a part. For example, if a part has two pins, *A\** and *B\_N*, and the *Read/Write* low-assertion character is defined as *\_N*, then pin *B\_N* is treated as a low-asserted pin on save.

#### *Additional Read*

Determines the pin name suffix — either *\** or *\_N* — that should be used in addition to the suffix specified in the *Read/Write* option to read low-asserted pins. For example, if a part has some low-asserted pins with *\** and some with *\_N* as suffixes in pin names, selecting *\** in the *Read/Write* option and *\_N* in the *Additional Read* option ensures that pins with both *\** and *\_N* in pin names are read as low-asserted pins.

**Note:** Even though Part Developer treats pin names with the suffix specified in the *Additional Read* as low-asserted pins, the pin text is not updated automatically. For example, suppose the *Read/Write* character is set to *\** and *Additional Read* to *\_N*, then all pins with *\_N* suffix will be read in as low-asserted and written with *\**. However, the

PIN\_TEXT property for these pins will continue to have the \_N suffix in them. This is with the assumption that to begin with the pin name was used as pin text.

**Note:** Selecting the *Use minus [-] sign for low asserted pins in Package view* option ensures that the pins that are considered low-asserted are written into the `chips.prt` file with a minus [-] sign. If you leave this unchecked, and depending on which pins you treat as low-asserted, they will get written into the `chips.prt` file either with a \* or \_N notation.

For example, consider a part with two pins, A\* and B\_N, with the \_N character configured to be the *Read/Write* low-assertion character. Now, if the *Use minus [-] sign for low asserted pins in Package view* option is selected, then on save the B\_N pin will get written as -B in the `chips.prt` file. If the option is unchecked, then the pin will get written as B\_N. In both the cases, the PIN\_TEXT property will have B\_N as the default value.

### ***Changing Low-Assertion Character for a Library***

If required, you can change the low-assertion character for existing library parts. However, to successfully change the assertion character it is recommended that you perform the following tasks.

- In Part Developer, enter the character — currently used as low-assertion character—in as the Additional Read character. In the Read/Write field enter the new low-assertion character.
- Create two libraries in this project; one containing the cells to be modified and other to store the modified cell.
- Run the con2con command. The syntax for the command is:

```
con2con -product PCB_Librarian_Expert -proj <project_file> -cdslib cds.lib  
-lib <oldlibrary> -cell <cellname> -outlib <library_with_updated_part>  
-outcell <newcellname>
```

Following migration steps updates the low assertion character information in the `symbol.css` file, thus ensuring that updated part can be used in the complete design flow.

### ***Example***

Modify a part, `mypart`, to change the low-assertion character from \_N to \*.

1. Launch Part Developer and create a new project, `myproj`.
2. In the Setup dialog box, from the Additional Read drop-down menu select \_N.
3. From the Read/Write drop-down list select \*.
4. Create two libraries; `oldlib` and `newlib`.

5. Copy mypart in oldlib.

6. Run con2con command.

```
con2con -product PCB_Librarian_Expert -proj myproj.cpm -cdslib cds.lib -lib oldlib -cell mypart -outlib newlib -outcell mypart
```

The updated part is available in the newlib library.

## Split Parts

A split part is a part where multiple symbols are created to represent a part. Since there are multiple symbols, there are special properties that go into the symbol and the `chips.prt` file ensuring that such parts can be used in the PCB design flow.

The entries in the *Split Parts* group box determines the values that go into the `chips.prt` and the `symbol.css` file when split parts are created using Part Developer. It has the following entries:

### Auto Add SWAP\_INFO to Chips

For large pin-count devices, it is possible that a single logical section is divided into multiple symbols. For example, an FPGA with 2000 input pins may have two symbols with 1000 pins each. Since both these symbols are part of the same logical function, it should be possible to swap an input pin from the first symbol with an input pin from the second. This is made possible with the `SWAP_INFO` property. This property goes into the `chips.prt` file and is used by `Packager_XL` and `PCB Editor`.

If this option is enabled, then when such a part is created in Part Developer and saved, Part Developer adds the `SWAP_INFO` property to the `chips.prt` file. The value for the `SWAP_INFO` property is determined by Part Developer on the basis of the `SPLIT_INST_GROUP` information provided through the [Edit Functions](#) dialog box. The functions with the same `SPLIT_INST_GROUP` value are combined into a single logical section. For more information about the `SWAP_INFO` property, see [Packager-XL Reference](#).

### Use SPLIT\_INST and \$LOCATION / Use SPLIT\_INST\_NAME

When creating split parts, select either the `SPLIT_INST` and `$LOCATION` properties or the `SPLIT_INST_NAME` property to be put on the split symbols. Either of these ensures that the symbol can be packaged into the same device and get netlisted as a single instance in the simulation netlist.

If you use the `SPLIT_INST` and `$LOCATION` properties, then assign the same value for the `$LOCATION` property on the symbol instances on the schematic that you want to combine into one package. For example, consider a large pin-count device, `MY_BGA`, which is split into four symbols. All the four symbols when instantiated on a design sheet in Allegro Design Entry HDL must have the `SPLIT_INST = TRUE` property and the same location property value, such as `LOCATION =ic1` on it. If there are two instances of the part, then you can use `$LOCATION = ic1` on one and `$LOCATION = ic2` on the other instance.

Similarly, if the `SPLIT_INST_NAME` property is used and the part is instantiated in Design Entry HDL, you need to ensure that the value of the property is same for all the split symbols, such as `SPLIT_INST_NAME = ic1`.

## Validation

The options in this group box determine whether validations will be run on a part when loading or saving a part. Additionally, you can determine the number of primitives on which to disable validations. By default, Part Developer will not validate parts with more than 20 primitives. This feature is helpful when loading large parts.

## Setting Up Package Defaults

To specify the values that Part Developer will use to create packages, click on the *Package* node. The following properties can be set up for a package:

### Class

Determines the part type. The possible part types are IC, IO, or DISCRETE. This information goes into the `chips.prt` file as the value of the `CLASS` property. By default, IC is selected.

### RefDes Prefix

Determines the reference designator prefix for the package. This information is added as a value of the `PHYS_DES_PREFIX` property in the `chips.prt` file. Packager-XL uses this property to write the prefix of the `LOCATION` property value (reference designator prefix). For example, if the reference designator prefix for a part is selected to be U and the part is used three times in a design, Packager-XL will assign the `LOCATION` values as U1, U2, and U3. The `PHYS_DES_PREFIX` property is typically used to group classes of parts such as resistors (R), capacitors (C), ICs (U), and inductors (L).

## Additional Package Properties

This grid is used to enter all other package properties. You need to enter the name of the property in the *Name* column and its value under the *Value* column. By default, the following commonly used properties are available through the Name drop-down list box:

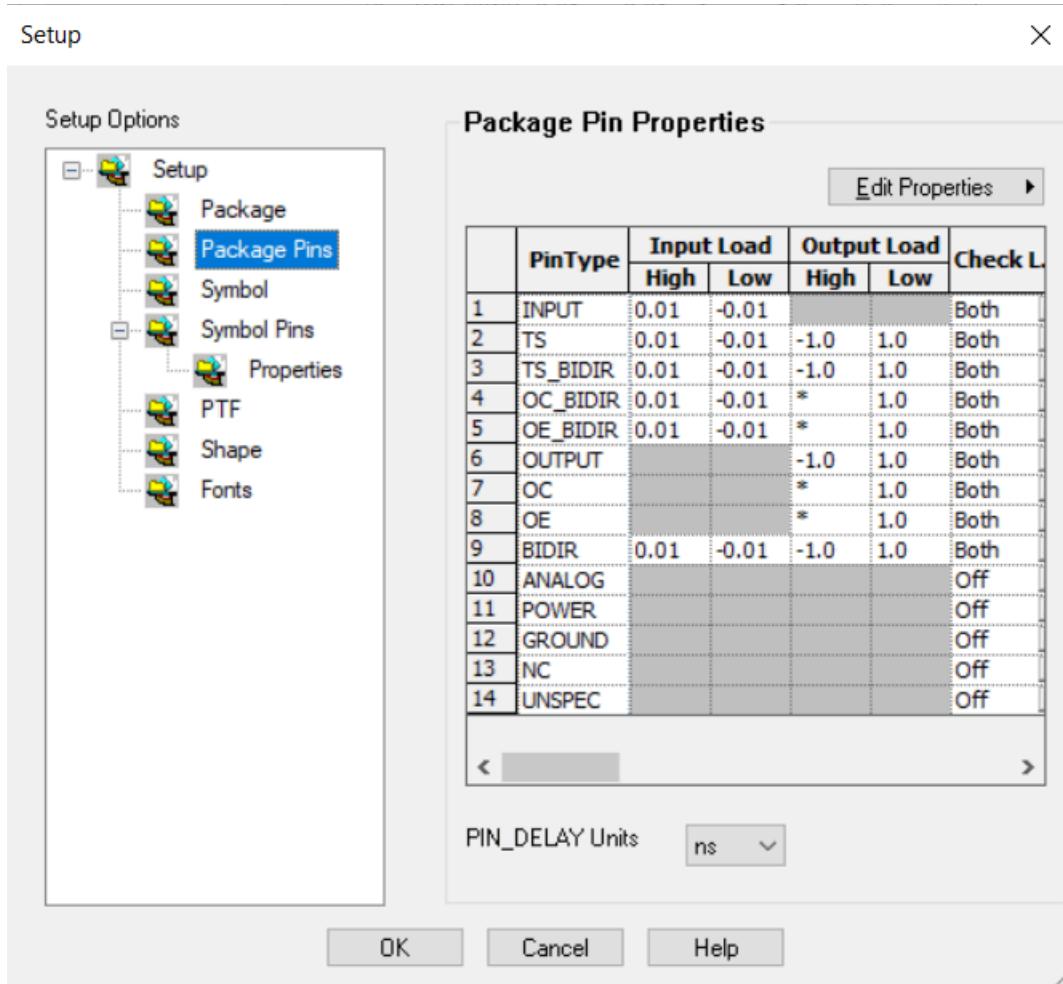
- *Family*
- *DEFAULT\_SIGNAL\_MODEL*
- *TECH*
- *POWER\_GROUP*
- *REF\_DES\_PATTERN*
- *DESCRIPTION*
- *BODY\_NAME*

## Setting Up Package Pin Properties

You can specify properties on a pin type basis through the *Package Pins* node. To set up the properties on the package pin types, click on the *Package Pins* node in the *Setup Options* tree. The right pane displays the *Package Pin Properties* group box.

# Part Developer User Guide

## Configuring Part Developer



The *Package Pin Properties* group box has the following entries:

- The *Edit Properties* button
  - A grid showing the pin types and the associated properties
  - A drop-down list box to specify the PIN\_DELAY units
- The possible values are nanoseconds and mils.

### Edit Properties

The *Edit Properties* button enables you to add, rename, or delete the properties that you have added to a pin. Whenever you add, delete, or rename a property, it gets added, deleted, or renamed in all the pin types by default. You can see the property and its value in the grid that shows the pin types and the associated properties.

To add a property to only a specific pin type, you need to first add the property and then delete the property values in the grid from the pin types to which the property should not be attached. Alternatively, add the property with null value and provide the value to only those pins where the property should be present.

For example, you want to add a property called `my_prop` with value `myvalue` to the input and output pin types. By default, when the property is added, it will be added to all the pin types. To remove them from all the pin types other than input and output, select a pin type, say `TS`, and delete the value of the `my_prop` property. This will ensure that the property `my_prop` is not associated with pin type `TS` in any package.

**Note:** By default, the `PIN_GROUP` property is available through the *Name* drop-down list. This property is used to determine the pins that are swappable with each other. For example, suppose a part has pins A, B, C, and D. If pins A and B can be swapped with each other and pins C and D can be swapped with each other, then the value of the `PIN_GROUP` property for pin A and B will be 1 and the value of `PIN_GROUP` property for pins C and D will be 2.

**Note:** You cannot delete the following predefined pin properties:

- *Input Load*
- *Output Load*
- *Load Checks*
- *PIN\_DELAY*

These are special properties and are used by different tools in the flow.

## Package Pins Grid

The *Package Pins* grid shows the following for the different pin types:

- *Input Load*
- *Output Load*
- *Load Checks*
- *PIN\_DELAY*

### Input Load

Determines the low and high input load values for the pin in mA. The default values for these fields are taken from *Setup*. The *Input Load* field is disabled for pin types where the input load values are not applicable.

## Output Load

Determines the low and high output load values for the pin in mA. The default values for these fields are taken from Setup. The *Output Load* field is disabled for pin types where the output load values are not applicable.

## Load Checks

Determines which Rules Checker checks will be run for a particular pin type. These are written as pin properties in the pin section of the `chips.prt` file. By default, all the options are selected. This implies that all checks will be run on all pin types. If you do not want to run a check on a specific pin type, clear the check box next to the pin type. For example, to disable the running of the `Dir` check on the `Input` pin type, deselect the `Dir` check box next to the `Input` pin type.

For more information about checks, see *Allegro Rules Checker User Guide*.

The following load checks are possible on a package pin:

### ***Check Load***

Determines how the `loading_check` Rules Checker rule is executed for a pin type. Setting the *Check Load* results in the `NO_LOAD_CHECK` property getting added to the particular pin type in the `chips.prt` file. This property is used by Rules Checker to execute the `loading_check` Rules Checker rule.

Depending on the option selected from the *Check Load* list, one of the following property-value pairs is added:

Option	Property in Chips.prt File
Off	<code>NO_LOAD_CHECK=BOTH</code>
High	<code>NO_LOAD_CHECK=LOW</code>
Low	<code>NO_LOAD_CHECK=HIGH</code>
Both	No property in chips file

The `loading_check` rule checks each signal (in high state and low state) for load violations. The rule checks the signals according to the value of the `NO_LOAD_CHECK` property. The checks are done in the following combinations:-

- In LOW state and NO\_LOAD\_CHECK=LOW, the pin will be ignored..
- In LOW state and NO\_LOAD\_CHECK=HIGH, the pin will be checked..
- In LOW state and NO\_LOAD\_CHECK=BOTH, the pin will be ignored..
- In HIGH state and NO\_LOAD\_CHECK=LOW, the pin will be checked..
- In HIGH state and NO\_LOAD\_CHECK=HIGH, the pin will be ignored..
- In HIGH state and NO\_LOAD\_CHECK=BOTH, the pin will be ignored.

### ***Check IO***

Determines how the `inputio_check` rule is executed for a pin. Setting the *Check IO* check results in the `NO_IO_CHECK` property getting added to the pin type in the `chips.prt` file. This property is used by Rules Checker to execute the `inputio_check` Rules Checker rule.

Depending on the option selected from *Check IO* list, one of the following property-value pairs is added:

Option	Property in <code>chips.prt</code> File
Off	<code>NO_IO_CHECK=BOTH</code>
High	<code>NO_IO_CHECK=LOW</code>
Low	<code>NO_IO_CHECK=HIGH</code>
Both	No property in <code>chips</code> file

The `inputio_check` rule checks that each signal is connected to at least two pins, and that at least one of the pins is an input pin. It also checks to ensure that a signal connected to a bidirectional pin is also connected to an input or output pin. The rule will check the net according to the value of the `NO_IO_CHECK` property:-

- In LOW state and `NO_IO_CHECK =LOW`, the pin will be ignored.
- In LOW state and `NO_IO_CHECK =HIGH`, the pin will be checked..
- In LOW state and `NO_IO_CHECK =BOTH`, the pin will be ignored..
- In HIGH state and `NO_IO_CHECK =LOW`, the pin will be checked.
- In HIGH state and `NO_IO_CHECK =HIGH`, the pin will be ignored.

- In HIGH state and NO\_IO\_CHECK =BOTH, the pin will be ignored.

#### ***Check Dir***

Determines if the pin is to have a direction check. The property written in the `chips.prt` file in case this option is not checked is `NO_DIR_CHECK=TRUE`.

#### ***Check Output***

Leaving the *Check Output* load check deselected results in the `ALLOW_CONNECT= FALSE` property to be added to the pin type in the `chips.prt` file. This property allows different types of outputs to be connected without producing errors when `OUTPUT_TYPE` properties are checked.

#### ***Check Assert***

Determines if the pin is to have an assertion check. The property written in the `chips.prt` file in case this option is not checked is `NO_ASSERT_CHECK=TRUE`.

#### ***Unknown Loading***

Selecting the *Unknown Loading* check box results in the `UNKNOWN_LOADING='TRUE'` property being written to the `chips.prt` file. This property turns off load checking for the pins.

### **PIN\_DELAY**

Specify the value of the `PIN_DELAY` property for a specific pin type. The unit of measurement can be specified through the *PIN\_DELAY Units* list box. By default, `ns` and `mil` are supported. The value needs to be specified as `<value> <units>`, such as `2 mil` or `2 ns`. Note that the `PIN_DELAY` value specified for a pin type will ensure that all packages that have the particular pin type will have the `PIN_DELAY` property associated with it. For example, if you put `2 ns` as the value of the `PIN_DELAY` for the `Input` pin type, the input pins for all packages and parts will get `PIN_DELAY` property with the value `2 ns`.

**Note:** You can also specify one of the following units of measurement for the `PIN_DELAY` property: `micron`, `microns`, `millimeters`, `mm`, `centimeters`, `cm`, `in`, `inches`, `meter`, `um`, `pm`, `nm`, `ps`, `us`, `ms`, `min`, `sec`, and `hour`. No conversions are done, and these values will be taken as is through the flow.

## Setting Up Symbol Properties

To specify the default values that Part Developer will use to create symbols, select the *Symbol* node. The attributes that define the look and feel of the symbols appears in the right pane. You can setup the following attributes:

### System Unit

Determines the unit of measurement for symbols. The default unit is inches. The possible units are fractions, inches and metrics.

### Sheet Size

Determines the Design Entry HDL schematic sheet size against which the symbols will be checked. An error is generated if the symbol size exceeds the schematic sheet size.

The *MAX\_SIZE* option in the drop-down list enables you to specify a maximum sheet size, which you can configure using the `Symbol_MaxSymSize` directive in your local or site CPM file. The default maximum sheet size specified in `setup.cpm` is `-18000,18000,18000, -18000`.

For information on how to add new sheet sizes to the *Sheet Size* drop-down list, see [Modifying the Sheet Size](#) on page 347.

### Pin grid size

Determines the spacing between the grids. The pins are placed on the symbol body only at grid points. The default grid size is 0.05 inches.

### Non-pin grid factor

Controls the density of the finer grid. The default value, 2, implies that the density of the finer grid is .025 of the other grid.

The density of the finer grid is derived in the following way:

$$\text{Finer Grid Density} = \text{Pin grid size value} / \text{Non-pin grid factor value}$$



When redefining the default values of *Pin grid size* and *Non-pin grid factor*, make sure that both values are multiples of .025.

### **Minimum Size - Height**

Determines the minimum height (in grids) for a symbol. For example, if *Height* has a value of 10 grids, the symbols will be at least 10 grids in height. In case the number of pins in a symbol does not fit into the minimum symbol height, the height will increase automatically as required. Therefore, small values for this field should be kept to ensure right-sized symbols.

### **Minimum Size - Width**

Determines the minimum width (in grids) for a symbol. For example, if *Width* has a value of 10 grids, then all the symbol that are created will be at least 10 grids wide. In case the number of pins in a symbol does not fit into the minimum symbol width, the width increases automatically as required. Therefore, small values for this field should be kept to ensure right-sized symbols.

### **Symbol Outline**

Determines the outline of the symbol that is being created. The possible values are *thick* and *thin*.

### **Auto Expand Bus**

This option is applicable for vector pins. Selecting this option results in the bits of the vector pin to be added as separate pins in the symbol. This enables you to add pin-specific properties to the bits of the vector bus.

**Note:** The *Auto Expand Bus* option is valid only when a symbol is created from the Symbol Editor. This option is not applied when creating symbols using the *Generate Symbol[s]* option.

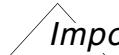
### **Text Attributes - Height**

Determines the height of any symbol text, such as the symbol name. The unit of measurement is inches.

### **Text Attributes - Color**

Determines the color of the symbol text. The possible colors are *MONO, RED, GREEN, BLUE, YELLOW, ORANGE, SALMON, VIOLET, BROWN, SKYBLUE, WHITE, PEACH, PINK, PURPLE, AQUA, and GRAY*.

The default value is MONO. This option ensures that the text is in contrast to the background color.

 **Important**

The Text Attributes options are disabled if the Enable Font Support option in the Fonts page is selected. With Enable Font Support option selected, the height and color of text attributes are read from the fields in the Fonts page.

### **Text Attributes - Rotation**

Determines the angle at which text appears on the symbol. Text can be displayed at 0, 90, 180, and 270 degrees.

### **Default Property Height**

Determines the height of the property names and values that appear on the symbol. The unit of measurement is inches.

 **Important**

This option is disabled if the Enable Font Support option in the Fonts page is selected. In this case, the height and color of text attributes are read from the fields in the Fonts page.

### **Symbol Properties**

*Symbol Properties* is a grid box in which you can specify properties, and property values and display attributes to be added to a symbol. The display attributes that you can set up are *Visibility*, *Color*, *Rotation*, *Height*, *Location*, and *Alignment*.

An example of a default property that you may want to put on every symbol can be your company name. To do so, you can specify a property called `Company_Name` and put the name of your company as its value. In case you want both the property name and its value to be visible, then select *both* from the *Visible* list. To display only the value of the property on the symbol, select the *Value* option.

## Setting Up Symbol Pin Defaults

The fields on the *Symbol Pins* panel determine the height, properties of pin text and pin attributes, such as pin spacing and stub lengths. The *Properties* node under *Symbol Pins* lets you specify the properties that can be assigned to symbol pins and symbol pin locations.

The following defaults are specified through *Symbol Pins*:

### Pin Name Height

Determines the height of the pin names in inches

### Use Pin Name as Pin Text

Determines whether pin names are used for pin text. If the option is selected, the `PIN_TEXT` property for a pin gets the pin name as its value. Otherwise, Part Developer does not put any value for the `PIN_TEXT` property. You will need to manually assign the `PIN_TEXT` property for each pin.

### Vector Bit Mask

Determines how a pin text is assigned to a vector pin such as `$Name`, `$Name LSB..MSB`, `$Name(LSB..MSB)`, `$Name[LSB..MSB]`, and so on. For example, if you have a vector pin A with 8 bits, selecting `$Name` displays the pin name as `A`. If you select `$Name LSB..MSB`, the pin name appears as `A 0..7`.

**Note:** The selection from the *Vector Bit Mask* drop-down list is applied only if the *Use Pin Name as Pin Text* option is selected.

### Pin Text Height

Determines the height of the pin text in inches.



This option is disabled if the Enable Font Support option in the Fonts page is selected.

### Pin Text Color

Determines the color of the pin text.



This option is disabled if the [Enable Font Support](#) option in the Fonts page is selected.

### Show Dot as Filled

Determines whether the hotspots on symbol pins are displayed as a filled circle.

### Minimum Pin Spacing

Determines the minimum spacing (in grid units) between the horizontally placed pins. For example, if the value in this field is 2 grids, then each horizontally placed pin will be at least 2 grids apart. In case after a symbol is created, you want to increase or decrease the pin spacing for particular pins, you can do so by using the *Move Pins* option on the Symbol Pins page of the Symbol Editor. See [Move Pins](#) on page 171 for more information.

### Low Assert Shape

Determines the shape of low-asserted pins. The possible shapes that a low-asserted pin can take is dot, line, and a line-dot combination.

### Stub Length

Determines the length of the symbol pin.

### Pin Name Format for Bus

Determines the format to be used for representing a vector pin. The vector pins can be displayed either in the <MSB..LSB> or <LSB..MSB> format.

## Setting Up Symbol Pin Properties

The *Properties* node enables you to specify the properties that you can put on symbol pins and determine the pin locations for the different pin types. It has the following fields:

## Symbol Pin Properties

Enables you to specify the properties that should appear on a symbol along with their values and display attributes. The display attributes that you can set up are *Visibility*, *Color*, *Alignment*, *Rotation*, and *Height*.

## Pin Location

Displays the pin types and their default locations on a symbol. For example, the default location for input pins is on the left of the symbol outline. This means that when you create a symbol using Part Developer, all input pins will appear to the left on a symbol. You can change the location of any of the supported pin types. The possible location values are left, right, top, and bottom.

You can also change the default location of any of the supported pin types by changing the value of `<symbol_pin_location>` parameter of the *PinType* directive in the `cds.cpm` file. For more information about the *PinType* directive, refer to the [\*PinType\*](#) section of Allegro Front-End CPM Directive Reference Guide.

## Setting Up PTF Defaults

The *PTF* option enables you to specify the default part table file properties for a part. The values that you can enter are the name, value, and context of the property. A PTF property and its value can be put in one of the following contexts:

- Key
- Global
- Injected
- Key and Injected

For more information about PTF properties and contexts, see the Part Table Editor User Guide.

## Setting Up Shape Defaults

The *Shape* option enables you to specify the default properties for a shape.

For more information about shape properties and how to set up a project for shapes, see the [Setting Up a Project for Shapes](#) section in the *Creating Shapes* chapter.

## Specifying Fonts

For displaying all text objects in Part Developer, you can either use vector fonts or fonts with ANSI character set. While using vector fonts you can only modify the color and size of the text objects.

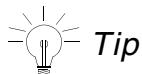
Using fonts with ANSI character set, enables you to specify different fonts and styles for different objects. The Fonts page of the Setup dialog box is used to customize the font used for displaying symbol text, symbol properties, reference designator, pin name, pin text, and pin properties.

UI Option	Description
Enable Font Support	Select this check box to enable support for fonts with ANSI character sets.  <b>Note:</b> This option is selected by default when you create a new project using Part Developer from release 16.3 onwards. If this check box is deselected, vector fonts are used.
Category	Select the symbol text object for which you want to set font and font attributes. You can set font attributes for 6 different categories of text objects.
Font	Select a font to display a specific category of text objects. The list displays all the fonts installed on the local system.  <b>Note:</b> On UNIX platforms, only the fonts installed in the mainWin folder are available in the Fonts drop-down list.

## Part Developer User Guide

### Configuring Part Developer

UI Option	Description
Size	Specify the font size in terms of fonts units. 72 Font unit equals an inch.   <b>Important</b> With the <u>Enable Font Support</u> check box selected, the values in the Size field is used as the default height for all text objects. Therefore, the text attribute fields – specifying the default values for the symbol text and pin text – in the Symbols page and the Symbol Pin page, are disabled. The values displayed in these fields are read from the Fonts page. In case of Height attribute, the value displayed in the Font size is converted in inches.
Style	Select a font style. The supported font styles are Regular, <b>Bold</b> , <b>BoldItalic</b> , and <i>Italic</i> .
Color	Select a color from a list of 15 colors: Aqua, Blue, Brown, Gray, Green, Orange, Peach, Pink, Purple, Red, Salmon, Skyblue, Violet, White, and Yellow.
Effects	Select an effect; either Regular or Underline.
Preview	Shows a preview of sample text after applying the selected fonts and font attributes.



If the Enable Font Support check box is not selected, the text height and colors for symbol text and Pin Properties are read from the corresponding fields in the Symbol page and the Symbol Pins page, respectively, of the Setup dialog box.

For all symbols created in Part Developer, the information about the size and the color of the text objects is saved with the symbol. The Font, Style, and Effects are used for display purposes and are therefore, based on the current project settings.

When symbols that use fonts with ANSI character set for displaying text objects, are used in designs that support only the vector fonts, the text objects are displayed using the default vector font. Only the size and the color of the text objects is as specified in the fonts page.

## Configuration

The *Configuration* option enables you to set up part construction rules and property setup. Then, using predefined configuration, you can quickly create parts that follow your company standards and yet have the flexibility to add properties and behaviors specific to the part.

### Creating New Configuration

You can save the following information as configuration in a `.tpl` file, which can be used as a template to create parts:

#### Packages

- Package properties
- Pin load properties

#### Symbols

- Grid size
- Symbol outline
- Pin text orientation
- Pin text size
- Whether to use pin names for pin text
- Minimum symbol height
- Minimum symbol width
- Symbol properties
- Symbol pin spacing, both top/bottom and left/right
- Position of symbol pin types

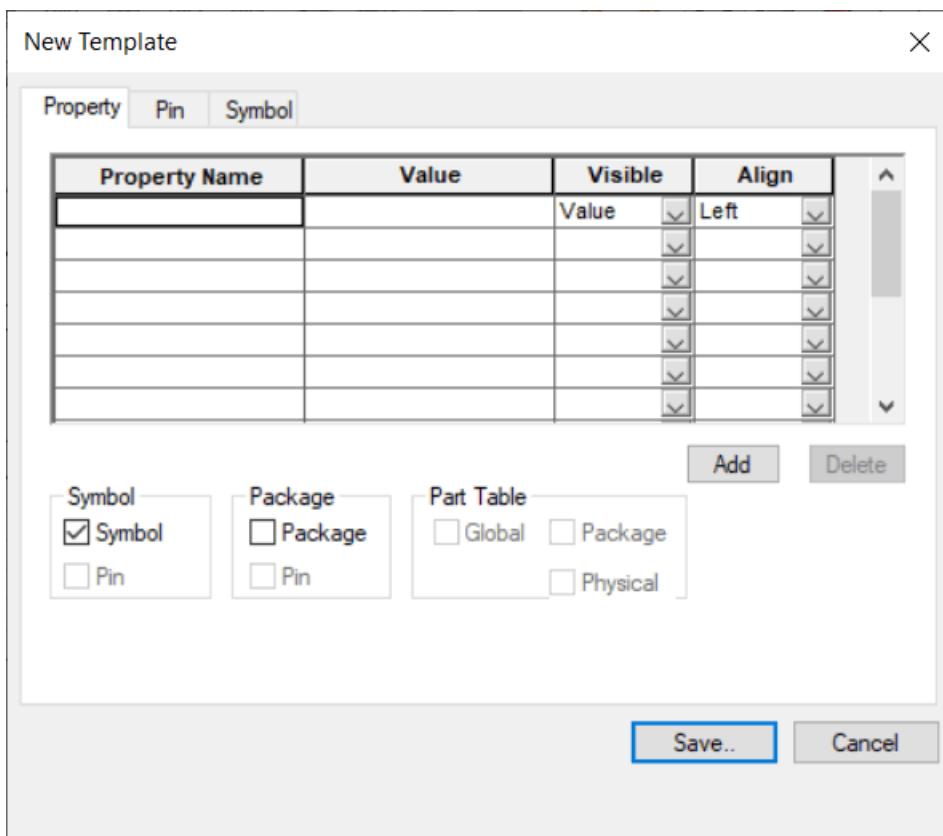
The steps are as follows:

## Part Developer User Guide

### Configuring Part Developer

1. Choose *Tools – Configuration – New*.

The New Template dialog box appears.



2. Enter the property name as `PART_NAME`.

3. Enter the property value as `?`.

This will ensure that the property is annotated to the symbol and the value for this property has to be specified by you in Design Entry HDL.

4. Select the value of the *Visible* field as `Both`.

5. Specify the alignment as `Center`.

6. Click the *Symbol* tab.

7. Specify the pin text size as `.6` grids.

8. Click *Save*.

9. Specify a filename.

**10. Click Save.**

This completes the creation of new configuration as a template. After you complete the creation, Part Developer asks you whether you want to apply the template in the current session. Choose *Yes* if you want to create parts based on the template in the current session of Part Developer. Otherwise, choose *No*.

## **Using Saved Configuration to Create Parts**

You can apply saved configuration to the current session of Part Developer and create one or more parts.

The steps to apply saved configuration to the current session of Part Developer are as follows:

- 1. Select *Tools – Configuration – Open*.**
- 2. Select the `.tpl` file that you want to load.**
- 3. Click *Open*.**
- 4. Click *Apply*.**

This applies the values stored in the part template to the current session of Part Developer.

In case you already have a part open, the loaded part will be updated for the following:

- Pin load values based on pin types
- Package properties, except for special properties that are not allowed in the Properties section, such as `PART_NAME`.

**Note:** To apply the template-related changes in symbols, you will need to regenerate them.

## **Verifying a Part Against a Template**

To ensure that the parts are conforming to your standards, Part Developer enables you to verify a part against a template. The verification is done only for those values that exist in the `.tpl` file. The output is displayed as a report that can be saved as a `.rep` file.

The steps to verify a part against a template are as follows:

- 1. Open the part.**
- 2. Choose *Tools – Verify*.**

3. Select the *Verify with Template* radio button.
4. Click *Verify*.
5. Browse and select the template against which you want to verify the part.
6. Choose *Open*.

The verification test report displays.

## Extracting Configuration from Existing Parts

If you have a part that has been built in compliance with your company standards, you can extract information from it and create a template. After the template information is extracted, the part is verified against the extracted template information. A verification report is generated listing the differences with the extracted values. Template information is extracted as per the following rules:

- [From Packages](#)
- [Pin Load Extraction](#)
- [Symbol Data Extraction](#)
- [Symbol Property Extraction](#)
- [Symbol Pin Extraction](#)
- [Grid Extraction](#)
- [Minimum Size Extraction](#)

### From Packages

- All properties found in any of the packages are added to the template with their values.
- If the property name matches with any of the properties listed below, the value is replaced with "?":
  - ❑ JEDEC\_TYPE
  - ❑ POWER\_PINS
  - ❑ NC\_PINS
  - ❑ BODY\_NAME
  - ❑ PART\_NAME

□ TECH FAMILY

### **Pin Load Extraction**

Pin load is extracted from the different pin types. If a pin type is not found in one of the packages, it is searched in the next package and so on. If a pin type is not found in any of the packages, its load is not added to the template. If for any pin type, the load extracted is not standard, i.e., the load is not same for all pins of a type in all packages, the first load is picked up.

### **Symbol Data Extraction**

- All symbols are read for a given part.
- All symbols must have at least one connection with a line stub or a bubble else an error stating that the process cannot proceed is displayed.

### **Symbol Property Extraction**

- All properties found in any of the symbols are added to the template with their values.
- If a property exists both in a package and a symbol, the package property is given precedence and its value is extracted.
- Alignment and visibility are extracted from the symbol and added.
- If the property differs in value across packages or symbols or it differs in visibility or alignment, the value of the property from the last instance of either the package or the symbol is extracted.

### **Symbol Pin Extraction**

For each pin, the following checks are done:

- A search is done for the first pin of each pin type and its location value is added to the template.
- The *Use Pin Names For Text* option is set to false if a single pin is found to violate this rule.
- The text style for pin notes is interpreted. If the style is vertical for top and bottom pins and horizontal for pins on left and right, the style is considered to be Automatic. The angles of 90 and 270 are considered equivalent and vertical and 0 and 180 are

considered equivalent and horizontal. If it is not consistent for all the instances for a pin type based location, the user is warned that this is not a standard.

## Grid Extraction

The highest common factor of all differences in X distances and Y distances is taken as the minimum grid unit. It is calculated into template in Inches units.

## Minimum Size Extraction

- The minimum pin spacing values on the left and right are read for all the symbols and the smallest value is extracted to the templates as the minimum spacing value for left and right. If in a symbol, 0 or 1 pin exist on left and right, its value is not extracted.
- The minimum pin spacing values at the top and bottom are read for all the symbols and the smallest value is extracted to the templates as the minimum spacing on top and bottom. If in a symbol, 0 or 1 pin exist on top and bottom, its value is not extracted.
- The symbol height value is read for all the symbols and the smallest value is extracted to the templates as the minimum symbol height.
- The symbol width value is read for all the symbols and the smallest value is extracted to the templates as the minimum symbol width.
- The outline is extracted as thick if all outlines are thick. The outline is extracted as thin if all outlines are thin. If neither of the cases is true, then no extraction is done for this value. If a value is not extracted, a warning is displayed.

To extract configuration in a `.tpl` file:

1. Load the part.
2. Choose *Tools – Configuration – Extract*.
3. Specify the location and the name for the template file.
4. Choose *Save*.

This creates a template with data extracted from the loaded part.

## **Creating Parts**

---

Parts usually correspond to physical objects in a PCB such as gates, chips, connectors, and so on that come in packages, such as DIP and SOIC. Normally, each of these packages will have one or more functions repeated one or more times. These functions are represented graphically through symbols. The symbols are used in Design Entry HDL, while the packages are used in Allegro PCB Editor.

In the HDL environment, a part is a collection of one or more of the following views:

- Package
- Symbol
- Simulation (mapfiles and wrappers)
- Part Table File

**Note:** For detailed explanations about the views of a part, see *Allegro Design Entry HDL Libraries Reference*.

You can easily create these views for a part using Part Developer. This chapter explains how to create a part from the ground up using Part Developer.

## **Part Types**

Parts can be classified into one of the following types:

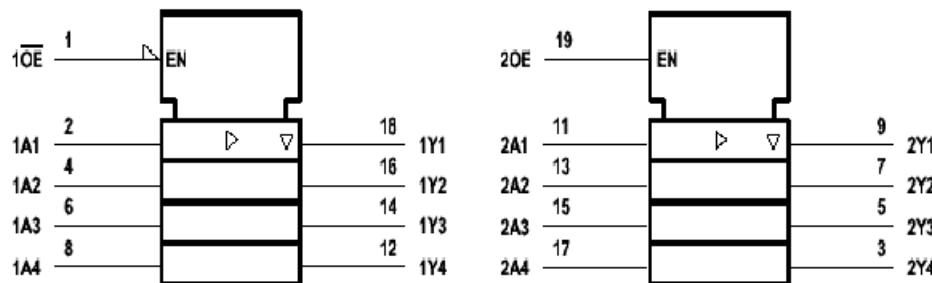
### **Symmetrical Parts**

A part that has only one logical function repeated one or more times in a package is called a symmetrical part. For example, LS00 with four independent 2-input NAND gates is a symmetrical part.

In the context of the `chips.prt` implementation, a symmetrical part will have the same logical pin list across all the slots. This implies that all logical pins are present in all slots of the part.

## Asymmetrical Parts

An asymmetrical part is one where multiple functions are present in a package. For example, LS241, an 8-slot part, with two different functionalities, is an asymmetrical part. The first four slots in such a package will have pin list A, Y, OE\*, VCC, and GND and the second four slots will have pin list A, Y, OE, VCC, and GND.



In the context of `chips.prt` implementation, an asymmetrical part has different pin lists across the slots. This implies that not all logical pins will be present in all the slots of the part. The slots in which a pin is not present are represented by 0 in the `chips.prt` file.

An asymmetrical part has multiple symbols, where each symbol represents one functionality. For example, LS241 will have two symbols, each representing one particular functionality.

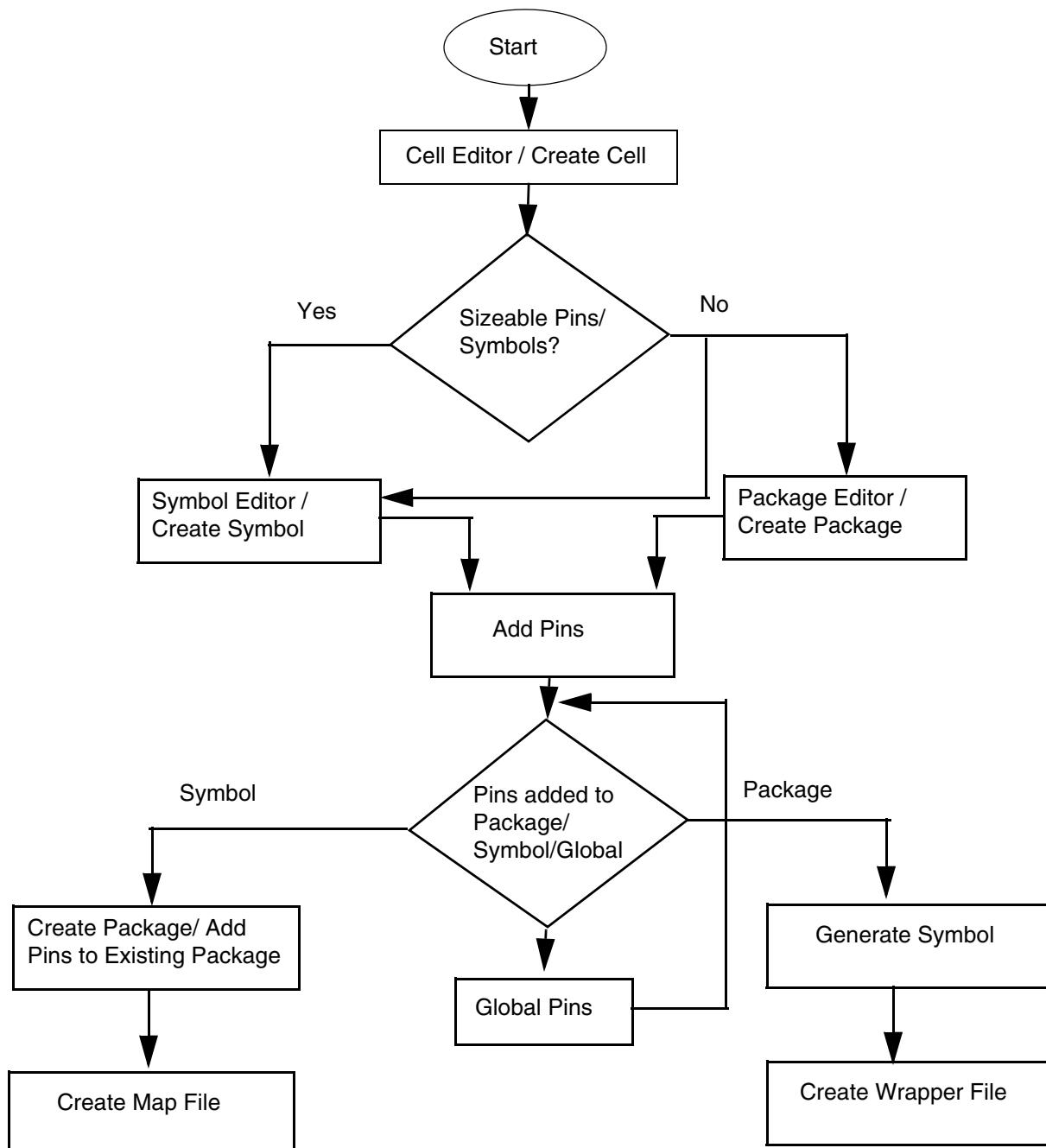
## Split Parts

A split part is a special case of an asymmetrical part. This part consists of a package in which logical pins are split across multiple slots. Split parts are useful for creating symbols for large pin-count devices. In a split part, each symbol represents a different functionality. The difference is that while in an asymmetrical part it is possible that a symbol represents multiple slots, a split part will have one symbol representing only one slot.

To create and use split parts, you need to add either the `SPLIT_INST` and `$LOCATION` or the `SPLIT_INST_NAME` properties on the symbol/chips. This can be done through *Setup*. For more information about these properties, see *PCB Systems Properties Reference*.

## Part Creation Methodology

The graphic below illustrates the methodology to be followed while creating parts:



As displayed, the part creation process begins with creating the new cell in a selected library followed by the logical pins entry. Logical pins entry can be done in one of the following ways:

- Through the Add Pin dialog box. This dialog box is accessed through the Package or Symbol Editor. You can add all the pins for a part through this dialog box.
- Add logical pins directly to a package through the *Logical Pins* grid of the Package Editor.
- Add logical pins directly to a symbol through the *Logical Pins* grid of the Symbol Editor.



**Important**

To make a sizeable symbol, you need to enter logical pins through the Add Pin dialog accessed through the Symbol Editor. This is required because **SIZE** is a pin-level property applicable on a symbol pin.

Once the pins are added, you create the packages/symbols. Packages can be created using the Package Editor. The Symbol Editor is used to create symbols. The wrappers and mapfiles can be created after the symbols and packages are created. The Verilog/VHDL map/wrapper file editors are used to create the wrappers and mapfiles.



The Cell Editor tree gives you the possibility to directly create a symbol from a package and vice-versa. You do not need to explicitly invoke the Package or Symbol Editor to create the packages or symbols. For more information, see [Direct Generation from a Symbol](#) on page 122 and [Direct Generation from a Package](#) on page 135.



**Caution**

***It is suggested that you go through Appendix D, “List of Valid Values in Part Developer” before you start creating parts.***

## Creating New Cells

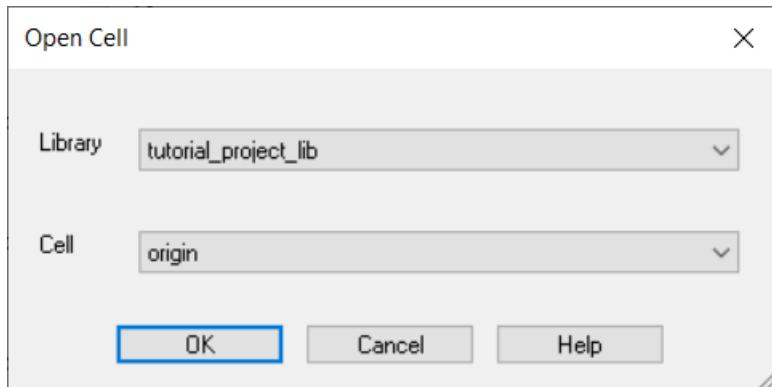
To create a new cell:

1. Choose *File – New – Cell*.

## Part Developer User Guide

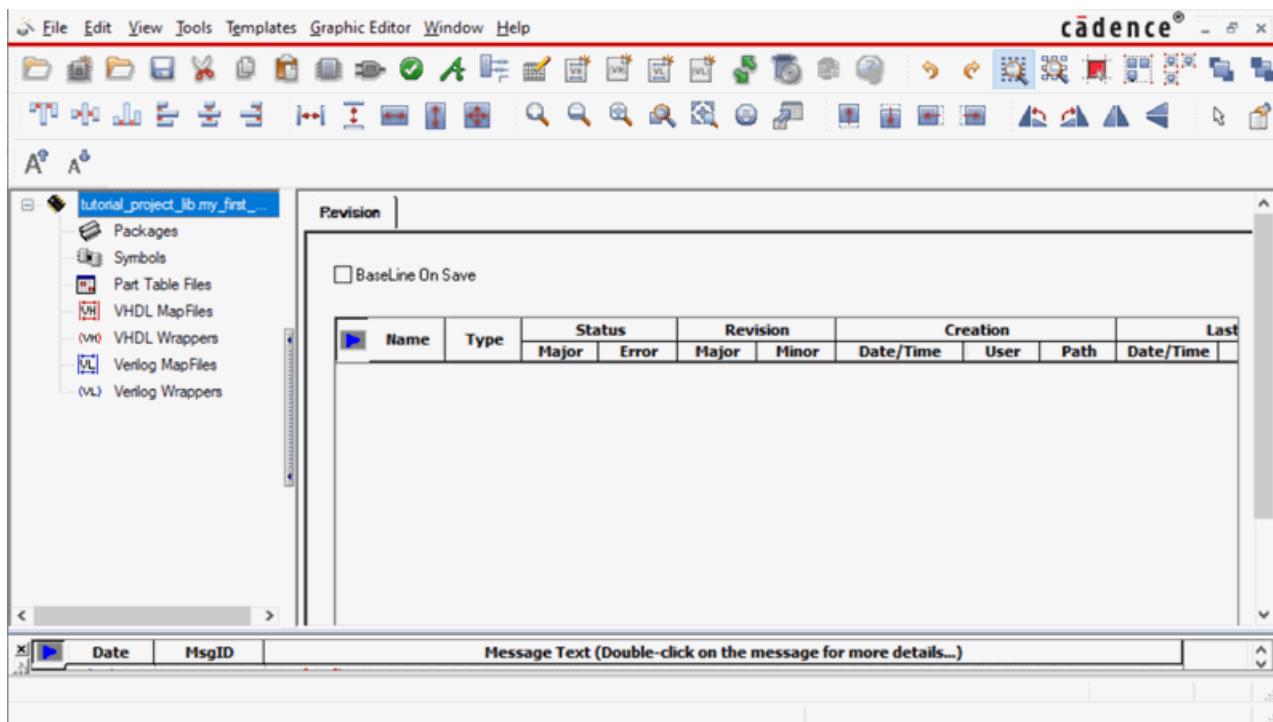
### Creating Parts

The New Cell dialog box appears.



2. Select the library in which to create the part through the *Library* drop-down list.
3. Specify the part name in the *Cell* field and click *OK*.

Part Developer displays the new part in the Cell Editor window. The *Session Log* shows the activity done by Part Developer to create the cell.



## Adding Logical Pins

Logical pins can be added in one of the following ways:

- Through the Add Pin dialog box. This dialog box can be accessed through either the Package or Symbol Editor. If you want to create a sizeable symbol, you must access the Add Pin dialog box through the Symbol Editor. See [Using the Add Pin Dialog Box](#) on page 112.
- Directly to a package through the *Logical Pins* grid of the Package Editor. See [Directly to a Package through the Package Editor](#) on page 117.
- Directly to a symbol through the *Logical Pins* grid of the Symbol Editor. See [Directly to a Symbol through the Symbol Editor](#) on page 119

Adding pins through the Add Pin dialog box method has the following benefits over the other two methods:

- Sizeable pins can be entered
- Vector pins can be entered at one step
- Pin names are validated for illegal characters
- Pin naming syntax is taken care of
- Pin information that is valid both in the context of packages and symbols can be specified. For example, both type (a package-level property) and location (a symbol-level property) can be specified for a pin.

## Using the Add Pin Dialog Box

### Through the Package Editor

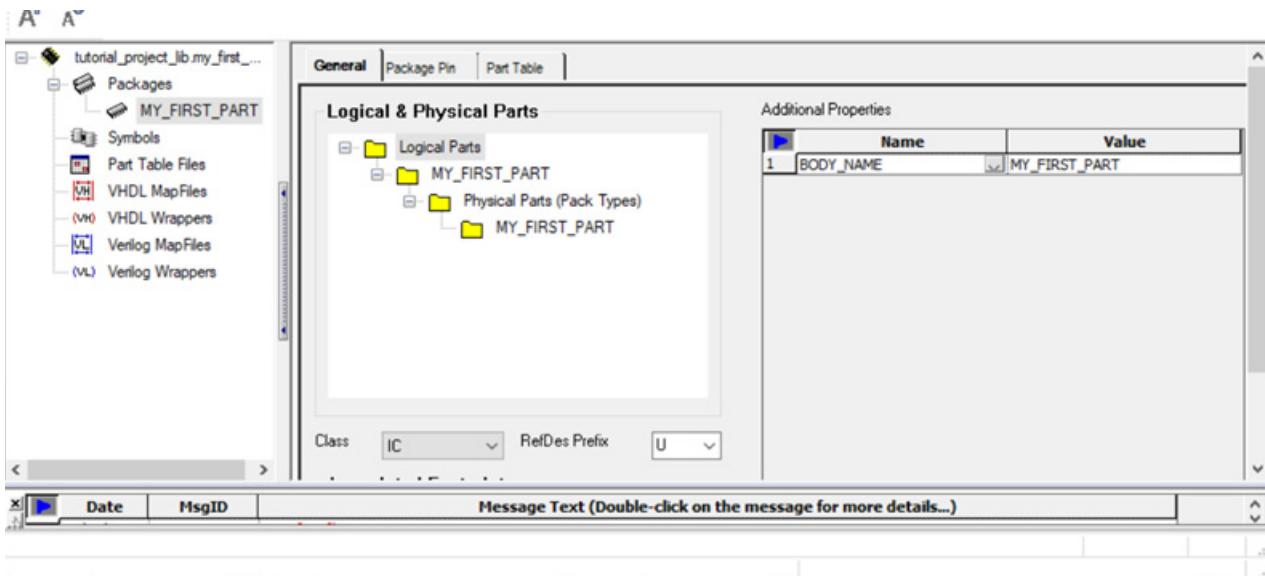
To use the Add Pin dialog box through the Package Editor:

1. Right-click on the *Packages* entry in the cell tree in the Cell Editor and select *New*.

## Part Developer User Guide

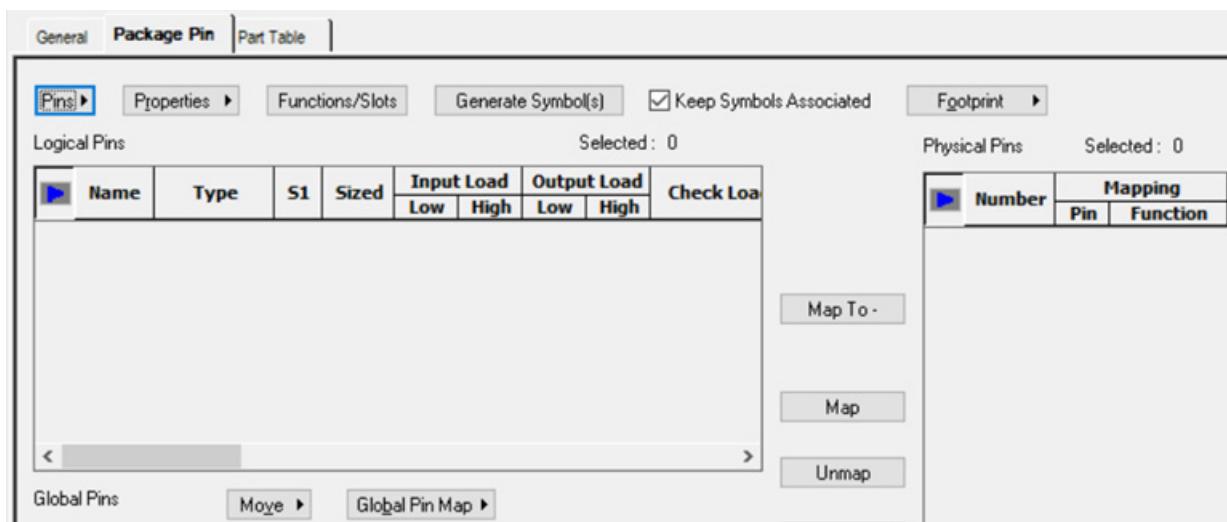
### Creating Parts

The Package Editor appears in the right pane of the Cell Editor. By default, the new package is created with the same name as the cell name.



#### 2. Click *Package Pin*.

The Package Pin page appears.



#### 3. Click *Pins – Add*.

The Add Pin dialog box appears. By default, the *Scalar* option is selected.

#### 4. Enter the logical pins as following:

- For scalar pins such as A, B, C, and D, specify A in the *From* field and D in the *To* field.
- For scalar pins such as A1–A10, specify A in the *Prefix* field, 1 in the *From* field, and 10 in the *To* field.
- For scalar pins such as A1Z - A10Z, specify A in the *Prefix* field, 1 in the *From* field, 10 in the *To* field, and Z in the *Suffix* field.
- For scalar low-asserted pins, specify the *Prefix*, *From*, and *To* values, and depending on what was selected as the low-assertion character on write, specify \* or \_N in the *Suffix* field.
- For vector pins, select *Vector*, specify the base name of the vector pin, the most significant bit (MSB), and the least significant bit (LSB). For example, to create a vector pin A<10..1>, enter A in the *Base Name* field, 10 in the *MSB* field, and 1 in the *LSB* field. If the pin is to be low-asserted, suffix the pin name with either \* or \_N character. The suffix character is determined by the *Read/Write* low-assertion character in *Setup*.

You can configure Part Developer to display vector pin bits in square brackets ([]) instead of angular brackets (<>). For more information, see [Configuring to Use Square Brackets in Vector Pin Names](#) on page 360.

**Note:** The *Sizeable* field is disabled when the Add Pin dialog box is accessed through the Package Editor. You can enter sizeable pins only when the Add Pin dialog box is accessed through the Symbol Editor.

5. Select the pin type from the *Type* drop-down list.
6. Specify the pin location from the *Location* drop-down list. By default, the pin location that was specified in the setup for the pin type is automatically selected.
7. Specify the pin load values through the fields in the *Load* group box. These fields are enabled only for those pin types for which load values are applicable, such as Input and OC. The default value for these fields are taken from the values specified in *Setup*.
8. Specify the checks through the fields in the *Check* group box. By default, the checks that were specified in *Setup* for the pin type are automatically selected.
9. Click *Add*.

The logical pins are added to the pin list at the bottom of the Add Pin dialog box.

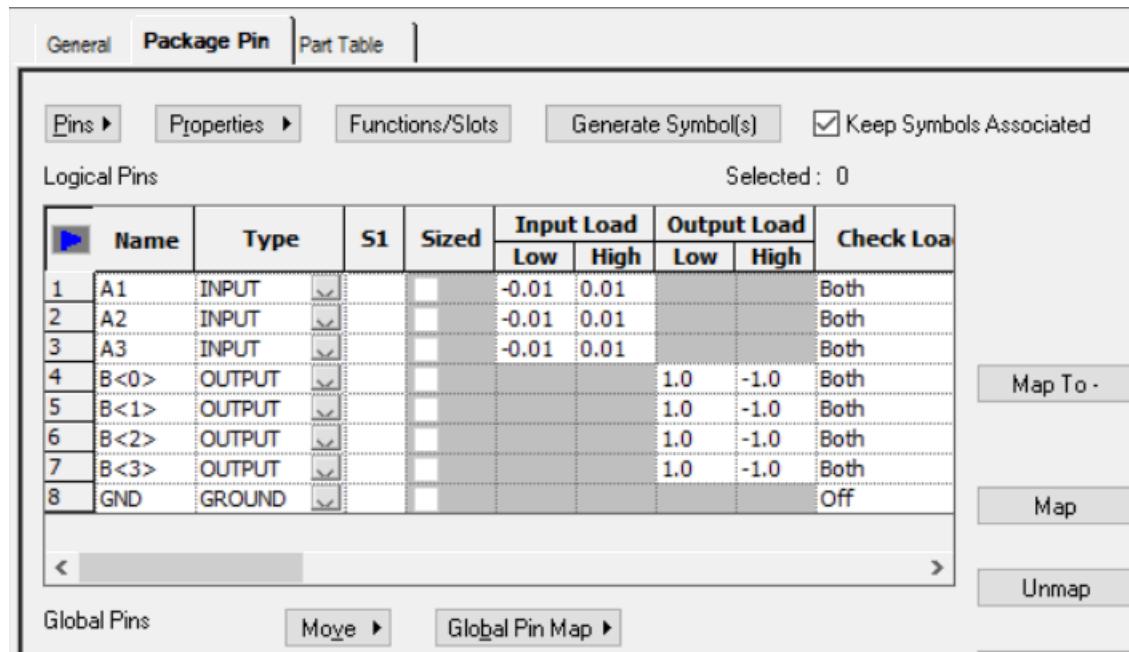
10. By default, all the pins that have been added are selected to be added to the package. Deselect the *Select* check box for pins not be added to the package.
11. Click *OK*.

## Part Developer User Guide

### Creating Parts

The logical pins are added to the package and appear in the *Logical Pins* grid.

**Note:** If global pins are added through the Add Pin dialog box, then the global pins (VCC, GROUND, and NC) are also added to the *Logical Pins* grid. If the global pins are not to be put in the symbol, move them down to the *Global Pins* grid by selecting the global pins and clicking *Move – Move Logical Pins to Global*.



### Through the Symbol Editor

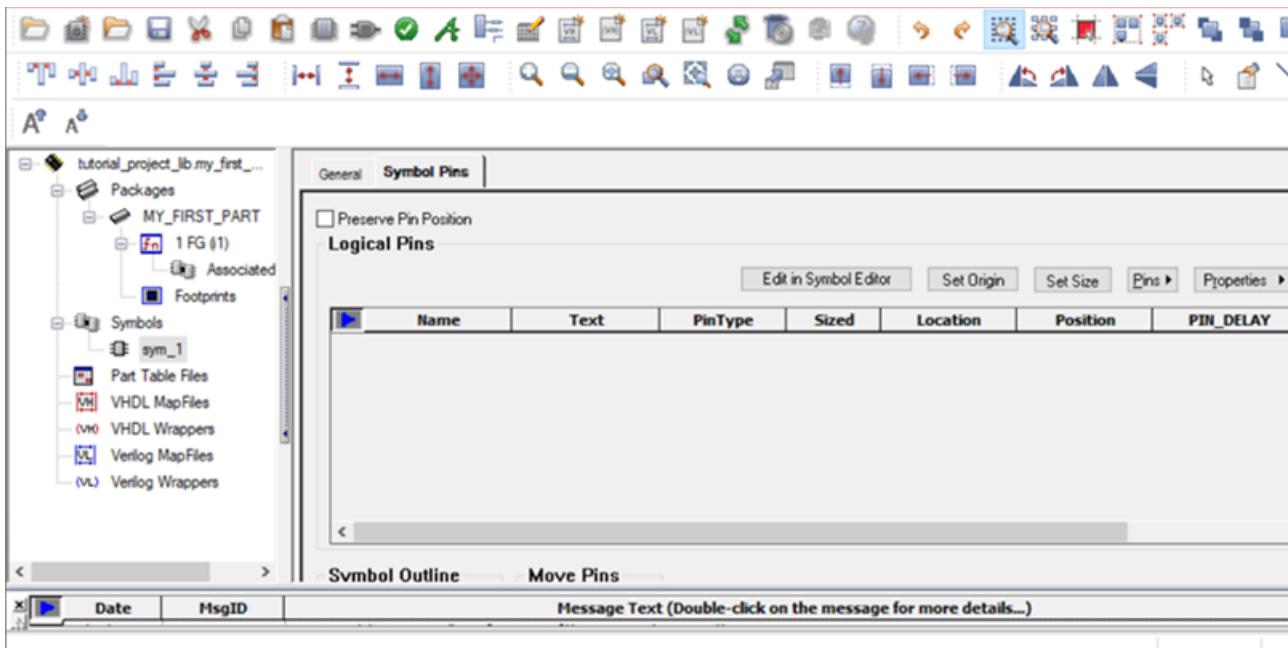
To use the Add Pin dialog box through the Symbol Editor:

1. Right-click on the *Symbols* entry in the cell tree in the Cell Editor and select *New*.

## Part Developer User Guide

### Creating Parts

The Symbol Editor appears in the right pane of the Cell Editor with a new symbol *sym\_1*.



#### 2. Click *Pins – Add*.

The Add Pin dialog box appears. By default, the *Scalar* option is selected.

#### 3. Enter the logical pins as following:

- ❑ For scalar and vector pins, use the same steps as discussed in using the Add Pin dialog box through the Package Editor.

**Note:** Depending on the *Auto Expand Bus* selection in *Setup*, vector pins are added as separate bits or as a single pin. For example, if the *Auto Expand Bus* option was selected, then adding a vector pin,  $A<3..1>$ , will add the pin to the symbol as  $A<1>$ ,  $A<2>$ , and  $A<3>$ .

- ❑ For sizeable pins, select *Sizeable* and enter the pin name in the *Base Name* field. The pin gets added with the *SIZE* property. For example, if a pin A is added as sizeable, then it gets added to the symbol as  $A<SIZE-1..0>$ .

#### 4. Select the pin type from the *Type* drop-down list.

#### 5. Specify the pin location from the *Location* drop-down list. By default, the pin location that was specified in the setup for the pin type is automatically selected.

#### 6. Specify the pin load values through the fields in the *Load* group box. These fields are enabled only for those pin types for which load values are applicable, such as Input and OC. The default values for these fields are taken from the values specified in Setup.

7. Specify the checks through the fields in the *Check* group box. By default, the checks that were specified in the setup for the pin type are automatically selected.

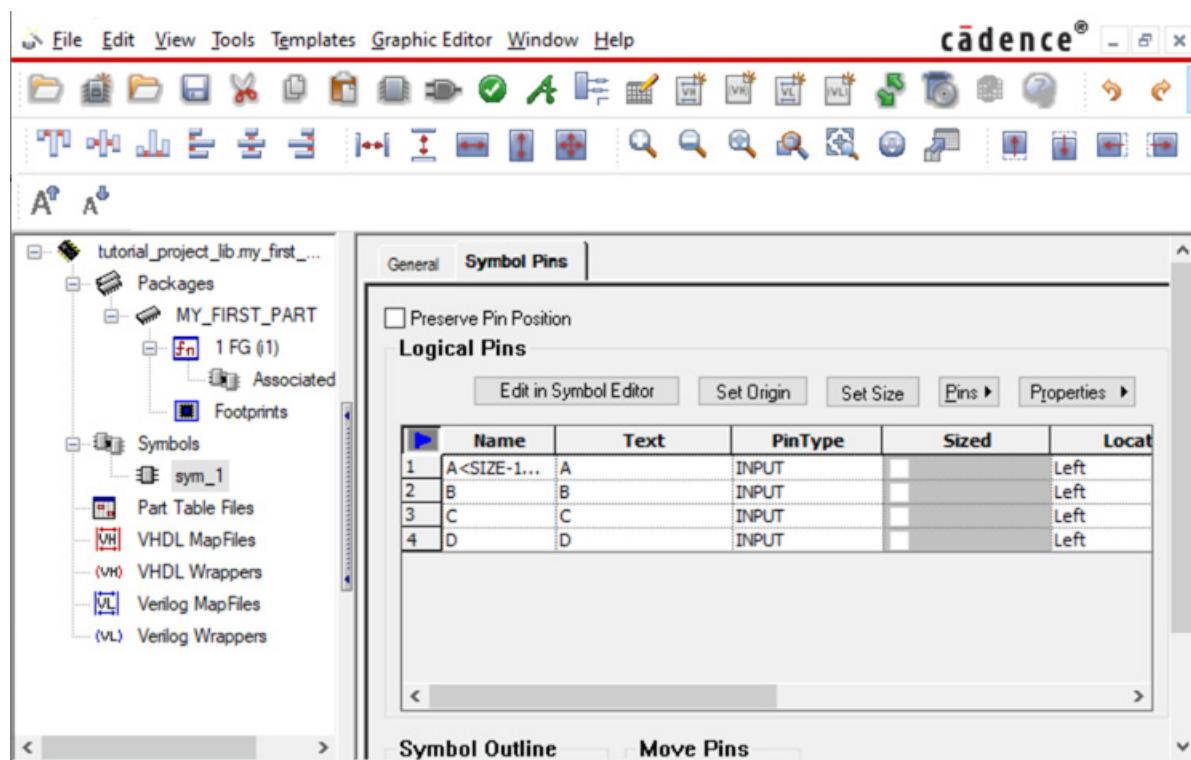
8. Click *Add*.

The logical pins are added to the pin list at the bottom of the Add Pin dialog box.

9. By default, all the pins that have been added are selected to be added to the symbol. Deselect the *Select* check box for pins not to be added to the symbol.

10. Click *OK*.

The logical pins are added to the symbol and appear in the *Logical Pins* grid.



## Directly to a Package through the Package Editor

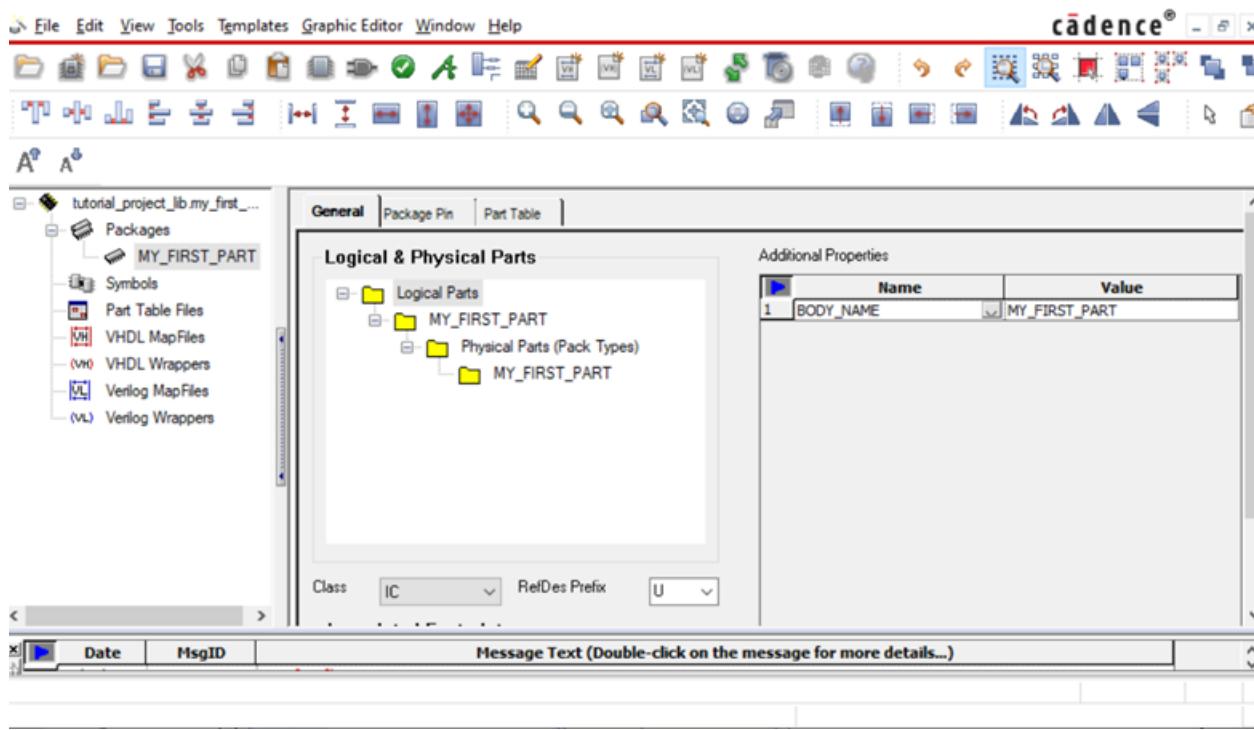
To enter the logical pins directly to a package through the Package Editor:

1. Right-click on *Packages* entry in the cell tree in the Cell Editor and select *New*.

# Part Developer User Guide

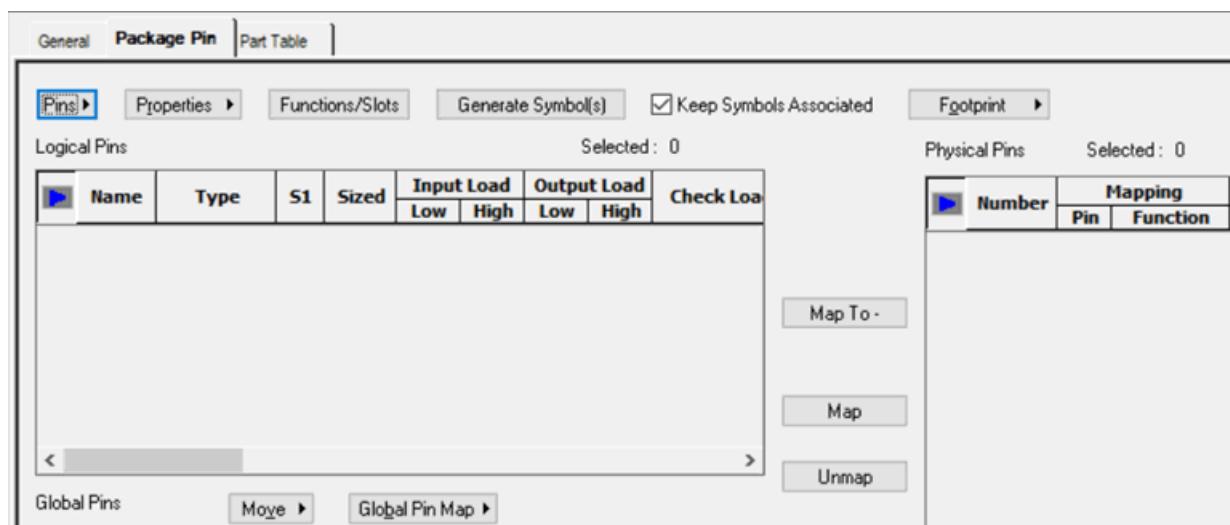
## Creating Parts

The Package Editor appears in the right pane of the Cell Editor. By default, the new package is created with the same name as the cell name.



### 2. Click *Package Pin*.

The Package Pin page appears.

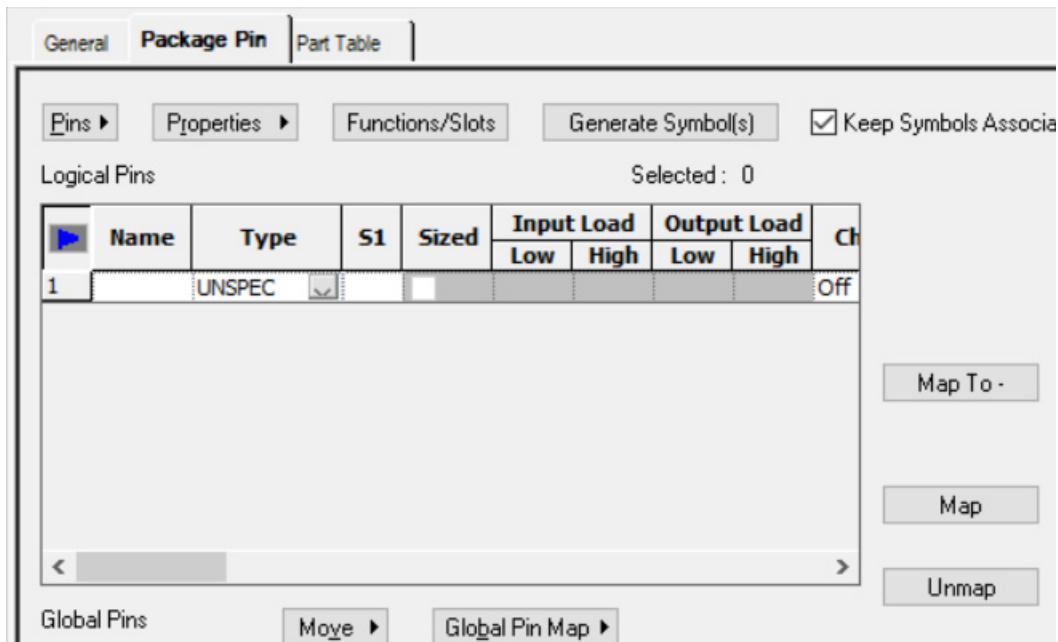


## Part Developer User Guide

### Creating Parts

3. Right-click in the *Logical Pins* grid and select *Insert Row After*.

A blank row gets added to the *Logical Pins* grid.



4. Enter the pin details, such as pin name, pin type, pin loads, and the checks to be performed on the pin.
5. Choose *File – Save* to save the pin information.

### Directly to a Symbol through the Symbol Editor

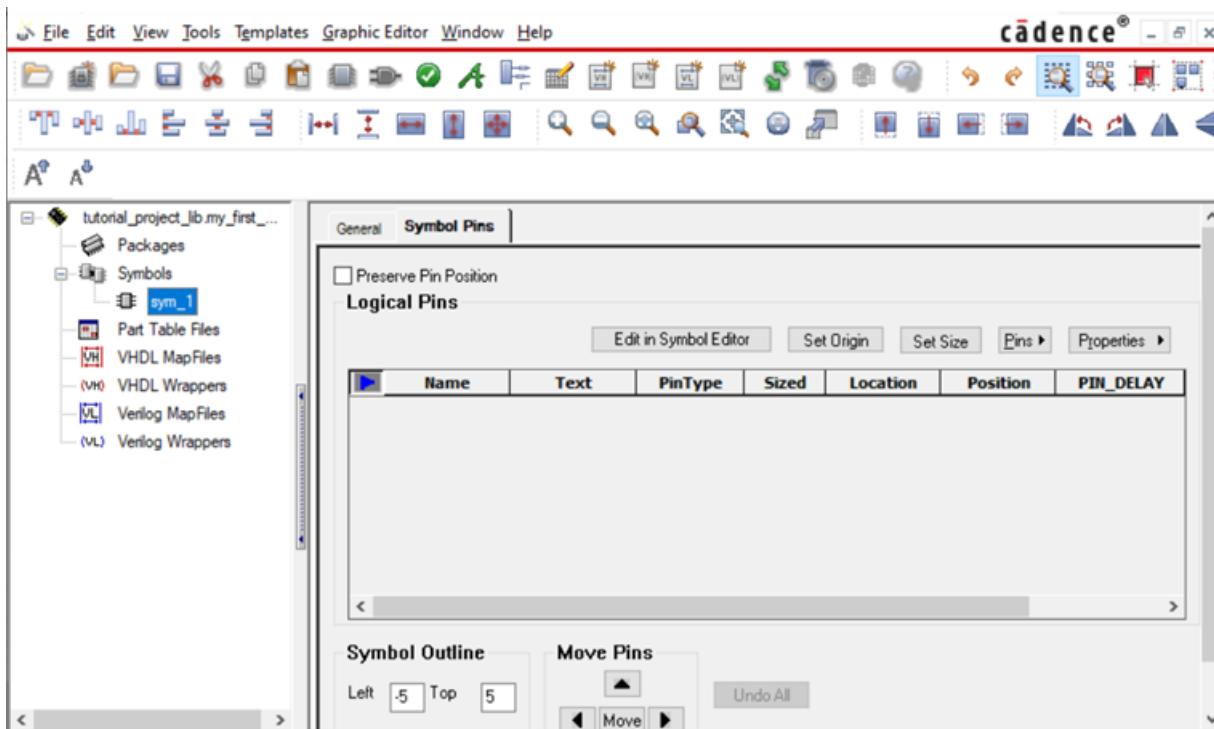
To enter the logical pins directly to a symbol through Symbol Editor:

1. Right-click on *Symbols* entry in the cell tree in the Cell Editor and select *New*.

## Part Developer User Guide

### Creating Parts

The Symbol Editor appears in the right pane of the Cell Editor with a new symbol *sym\_1*.



2. Right-click in the *Logical Pins* grid and select *Insert Row After*.

A blank row gets added to the *Logical Pins* grid.

3. Enter the pin details, such as pin name, pin text, pin location, and pin position.

**Note:** Pin type cannot be specified when a pin is entered directly through the Symbol Editor. This is because pin type is a package-level information and not required in the context of a symbol.

## Adding Global Pins

Global pins (VCC, GROUND, and NC) can be added through either the *Add Pin* dialog box or through the Package and Symbol Editor. If the pins are added through the *Add Pin* dialog box, they are added to the *Logical Pins* grid. This implies that these pins will appear in the symbols as well. If you do not want the global pins to be added in the symbols, then add them to the *Global Pins* grid. To add pins to the *Global Pins* grid, do the following:

1. Press **Ctrl + I** in the *Global Pins* grid.

A blank row appears in the *Global Pins* grid.

2. Enter the global pin name and type in the Name and Type columns.

**Note:** NC pins must necessarily be named NC. Otherwise, Part Developer will report errors.

For information on moving pins, see [Modifying Pin Lists and Mapping](#).

## Setting and Retrieving Pin Order

You can rearrange the pins in the *Logical Pins* grid according to your requirements and set the pin order. Part Developer saves the information as metadata and enables you to retrieve the set pin order.

Pin information is displayed in the Distribute Pins dialog box in the same order as in the Logical Pins grid. Therefore, if the order in which pins are to be distributed for a split part is set, you can retrieve the pin order when you want to create the split part.

### Setting Pin Order

Setting pin order saves the order in which logical pins are displayed in the *Logical Pins* grid of the Package Editor.

To set pin order:

- Choose *Pins – Set Pin Order*.

The pin order for the package is saved.

### Retrieving Pin Order

Retrieving pin order displays the pins in the *Logical Pins* grid in a preset order.

To retrieve pin order:

- Choose *Pins – Retrieve Pin Order*.

The logical pins are rearranged in the stored order.

## Creating Packages

A package contains the physical information about a part. The following information needs to be specified when creating a package:

- The package name
- Equivalent packages. Equivalent packages are those that share the same logical and physical pin list and properties.
- The footprint information (JEDEC\_TYPE and ALT\_SYMBOLS)
- The class of the part
- The reference designator prefix
- Package properties
- Logical pins in the package
- Number of slots in the package
- The physical to logical pin mapping
- The package pin properties

Part Developer provides two ways to create packages:

- By using the Package Editor. See [Using the Symbol Editor](#) on page 135.
- Directly from a symbol. See [Entering Symbol Information](#) on page 135.

## Using the Package Editor

To create packages using the Package Editor, right-click on the *Packages* entry in the Cell Editor and select *New*.

## Direct Generation from a Symbol

To directly generate a package from a symbol, do the following:

1. Right-click on a symbol in the Cell Editor and select *Generate Package*.

In both of the above-mentioned ways, the Package Editor appears in the right pane of the Cell Editor. If there are no existing packages, the new package is created with the same name as the cell name. If packages already exist for the part, then the new package is created as `<cell_name>_n`. For example, consider the part `LS00` with an existing package `DIP`. When a new package is created, it will appear in the cell tree as `LS00_1`. The package name can be changed as required.

## Entering Package Information

The information required to successfully complete the package creation is divided between the General and Package Pin page. The Package Editor appears with the *General* page selected by default.

The following information is specified through the General page:

- The logical and the physical parts for a package
- CLASS property
- PHYS\_DES\_PREFIX property
- Footprint information
- Any other package-specific properties

The Package Pin page enables you to enter the pin information about a package. The logical, physical pins and the mapping between them is done through this page. For more information about the fields of these two pages, see [Package Editor](#) on page 44.

You can begin with either the General or Package Pin page. In this guide, information is specified on the General page followed by the Package Pin page.

1. When a package is created, the *Logical & Physical Parts* tree shows the name of the package as the logical part and the physical part. To create a new logical part, right-click on *Logical Parts* and select *New*.

The Add Logical Part dialog box appears.

2. Enter the logical part name and click *OK*.

The logical part appears as a node under the *Logical Parts* node. By default, the *Physical Part(Pack Types)* node with a physical part with the name same as the logical part name appears under the logical part name node. This physical part is the default pack type for the logical part.

3. To create a new physical part (pack type), right-click on the *Physical Parts(Pack Types)* node and select *New*.

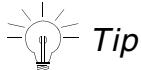
The Add Physical Part dialog box appears.

4. Enter the pack type in the *Pack Type* field and click *OK*. For example, to create a DIP package, enter DIP.

The physical part appears under the *Physical Parts(Pack Types)* node. For example, if the logical part was LS00 and you created a physical part DIP, the LS00\_DIP will appear under the *Physical Parts(Pack Types)* node.

5. Specify the class of the package. This can be done by selecting a value from the *Class* drop-down list. The possible values are IC, IO, or DISCRETE. This information goes into the `chips.prt` file as the value of the CLASS property.
6. Specify the reference designator prefix. This is done by either selecting a value from the *RefDes Prefix* drop-down list or entering a new value into the *RefDes Prefix* field. This information goes into the `body` section of the primitive as the value of the PHYS\_DES\_PREFIX property.
7. Specify the JEDEC\_TYPE property. To do so, click on the browse button next to the *Jedec Type* field.

The Browse Jedec Type dialog box appears. This dialog box shows all the available footprints.



Part Developer provides a filter using which only those rows that match the filter conditions are shown. To set the filter condition, do the following:

- a. Right-click on any of the rows and select *Filter Rows*.

The Filter Rows dialog box appears.

- b. Specify the filter condition and click *OK*. For example, to display only DIP footprints, specify `dip*` as the filter condition.

The Browse Jedec Type dialog box will show only the DIP footprints.

8. Select a footprint and click *OK*.
9. Specify the Alt\_Symbols property. To do so, click on the browse button next to the *Alt Symbols* field.

The Browse Alt Symbol dialog box appears. This dialog box shows all the available footprints.

10. Select a footprint and click *OK*.
11. Enter additional properties. To do so, right click in the *Additional Properties* section and select *Insert Row After*.

An empty row appears in the *Additional Properties* section.

- 12.** Specify the property name and value in the *Name* and *Value* columns.

Next, the package pin information needs to be specified. To do so:

- 13.** Click *Package Pin*.

The Package Pin page appears. The *Logical Pins* grid shows all the logical pins in the package. You need to add the physical pin numbers and slots and do the mapping between the logical and physical pins.

Physical pins can be added either manually or by extracting from a footprint.

- 14.** To add physical pins manually, click *Footprint – Add Physical Pins Manually*.

The Add Physical Pin Numbers dialog box appears.

- 15.** Enter the physical pin numbers in the Add Physical Pin Numbers dialog box. The physical pins can either be linear or alphanumeric as in CGA or BGA. By default, the *Linear* option is selected. If you want to add alphanumeric pin numbers, choose the *Grid* radio button.

**Note:** In the *Grid* option, enter the row column labels. The input format of these fields is the same as that for the numeric field except that it can have alphanumeric ranges also. The pin numbers generated are row major in nature. For example, if the *Row Labels* is A, B and so on and the *Column Labels* is 1-5, then the pin numbers are A1, A2, A3, A4, A5, B1, B2, B3, B4, and B5.

- 16.** Click *OK*.

The pin numbers appear under the *Physical Pins* grid.

- 17.** Alternatively, to get physical pin numbers from the specified footprint, choose *Footprint – Extract From Footprint*. The *.dra* files are used to extracting the pin numbers from the footprint. However, the *.psm* file is used to display the footprint information in the Footprint Viewer.

**Note:** In case no footprint has been specified on the *General* page, choose *Footprint – Select & Extract from Footprint*. This will enable you to first select a footprint and then extract the physical pin list from the selected footprint.

Next, you need to specify the slot information.

- 18.** By default, Part Developer creates the packages with one slot. To create more slots, click *Functions/Slots*.

The Functions/Slots dialog box appears.

- 19.** To add slots to the package, click *Add*.

A new slot is added to the package.

- 20.** After the required number of slots have been added, determine which logical pins are present in the slots. To do so, click *Distribute Pins*.

The Distribute Pins dialog box appears. By default, Part Developer puts all the logical pins in the first slot. Select the slots in which the logical pins need to be present and click *Ok*.

The slots that were added appear in the Logical Pins grid. The slots in which the logical pins are not present is marked by ‘-’.

- 21.** Add the global (GROUND, NC, POWER) pins that the package may have in the *Global Pins* grid. To do so, right-click in the *Global Pins* grid and select *Insert Row After*.

An empty row appears in the Global Pins grid.

- 22.** Enter the pin name and determine the pin type from the *Type* drop-down list.

After entering the logical pins, you need to do the mapping of the logical pins with the physical pins.

- 23.** To do the logical-to-physical pin mappings, select the logical pins and the slots for which mapping is to be done. Next, select an equal number of pins from the *Physical Pins* list and click *Map*. The mapping is done as per the order in which the logical and physical pins were selected. For example, if for logical pin A, slot S1, S3, and S2 were selected in that order and the physical pins selected were A3, A2, and A1, A3 will get mapped to slot S1, A2 to slot S3, and A1 to slot S2.

- 24.** Next, map the physical pins to the global pins. You can map one global pin to more than one physical pin. For example, suppose a package has pin numbers 1, 3, and 5 as three ground pins. To do such a mapping, select the *Global Pins* row that has GND as its entry, select the pin numbers 1, 3, and 5 from the *Physical Pins* list and click *Map*.

**Note:** Only one global pin can be mapped to one or more physical pins at a time.

An alternative method is by using the *Global Pin Map – Edit* option. See [Global Pin Map – Edit](#) on page 133 for more details.

This completes the process of creating a package.

## Specifying PIN\_DELAY

Pin delay values are attached to physical pins starting from 15.5.1 Release. This is required only for multi-section parts as each logical pin in such parts is mapped to multiple physical pins. In the case of single-section parts or split parts, the pin number is redundant as one logical pin in these parts is mapped to a single physical pin.

The syntax for specifying `PIN_DELAY` for multi-section parts is as follows:

```
PIN_DELAY = '(PinNum1:val1;PinNum2:val2 mil;PinNum4:val4)';
```

In the above syntax, `val` includes the unit also.

The following syntax is valid for specifying `PIN_DELAY` in the case of single-section parts and split parts:

```
PIN_DELAY = '(value)';
```

**Note:** For multi-section parts, the `PIN_DELAY` value must be specified in the correct syntax.

The `PIN_DELAY` property can be specified by doing the following:

1. Select the pin for which you want to specify the `PIN_DELAY` in the Logical Pins grid of the Package Editor.
2. Scroll to the extreme right.

The `PIN_DELAY` column will appear.

3. Enter the value of the `PIN_DELAY` property as required. By default, the unit of measurement selected in *Tools – Setup – Package Pins – PIN\_DELAY Units* will be used as the unit of measurement. For example, if mil was selected in Setup and you enter 2 in the `PIN_DELAY` column, Part Developer will automatically make the property value as `2 mil`. In case you want to enter any other unit of measurement, you need to explicitly specify that. The supported values are micron, microns, millimeters, mm, centimeters, cm, in, inches, meter, um, pm, nm, ps, us, ms, min, sec, and hour.

## Creating Split Parts

To create a split part, do the following:

1. Do the logical-physical pin mapping for the first slot.
2. Create the necessary slots.
3. Distribute the pins across the slots.

**Note:** The advantages of the above method are:

- It is easy to do mappings for a single slot.
- After distributing the pins, the slots that are left unmapped are automatically marked as -.

## Adding Package Pin Properties

To add package pin properties, do the following:

1. Select the Package Pin page.
2. Choose *Properties – Add*.

The Add Properties dialog box appears.

3. Specify the name of the property in the *Name* column and its value in the *Value* column and click OK.

**Note:** By default, the `PIN_GROUP` property is available through the *Name* drop-down list. This property is used to determine the pins that are swappable with each other. For example, if a part has pins A, B, C, and D, and pins A and B can be swapped with each other, and pins C and D can be swapped with each other, then the value of the `PIN_GROUP` property for pins A and B will be 1 and the value of the `PIN_GROUP` property for pins C and D will be 2.

**Note:** By default, the properties are added to all the pins of the package. To disassociate a property from a particular pin, delete the property value. A null value implies the property is absent from the pin.

## Adding Differential Pair Properties

During part creation, librarians can capture differential pair information from datasheets. This support for library-defined differential pairs enables the differential pair property to be stored in verified part libraries, which are available to all designs.

Part Developer uses the values specified in `DIFF_PAIR_PINS_POS` and `DIFF_PAIR_PINS_NEG` properties to identify a set of two pins as a differential pair and to designate the positive and negative pins. The `DIFF_PAIR_PINS_POS` and `DIFF_PAIR_PINS_NEG` properties are stored in the following syntax:

```
DIFF_PAIR_PINS_POS='diff_pair_name';
DIFF_PAIR_PINS_NEG='diff_pair_name';
```

**Note:** Two pins with the same `diff_pair_name` value constitute a differential pair.

## Points to Remember about Differential Pair Support in Part Developer

- The differential pair property is supported only at the package level and is saved in chips only.
- The differential pair property is associated with logical pins.

- The positive and negative pins comprising a differential pair must have the same pin type.
- The differential pair property cannot be associated with GROUND, POWER, and NC pin types.

### Procedures for Creating and Deleting Differential Pairs

- To add differential pair properties to all parts in a library, you can run the `con2con` command from the command prompt. For information on the syntax of the `con2con` command, see [con2con](#) on page 370.
- To add differential pair properties to any part through the Package Editor based on predefined rules in `cds.cpm`, see [Autocreating Differential Pairs through the Package Editor](#) on page 129.
- To remove differential pair properties, see [Deleting Differential Pair Properties](#) on page 132.

### Autocreating Differential Pairs through the Package Editor

When you autocreate differential pairs, Part Developer adds differential pin properties to the pins that match any of the naming schemes specified in the `DiffPair_Recognition_Rules` definition in `cds.cpm`, located at `<your_inst_dir>/share/cdssetup/projmgr`. You can specify whether you want to create differential pairs for a selected package or all packages.

 *Important*

Each time you run *Auto Create Differential Pairs*, only the pins that match the naming schemes specified through the `DiffPair_Recognition_Rules` directive are created as differential pairs. Existing differential pins are not modified even if these pins do not match the naming schemes in the current `DiffPair_Recognition_Rules` definition.

To autocreate differential pairs through the Package Editor, do the following:

1. Configure the default `DiffPair_Recognition_Rules` definition in `cds.cpm` if required.  
For information on the `DiffPair_Recognition_Rules` definition, see [Specifying a Naming Convention for Autocreation of Differential Pairs](#) on page 130.
2. Configure the default `Default_Diffpair_Value` definition in `cds.cpm` if required.

For information on the `Default_Diffpair_Value` definition, see [Naming Differential Pairs](#) on page 131.

3. Right-click in the *Logical Pins* grid.
4. Choose the *Auto Create Differential Pairs* menu item from the shortcut menu.
  - a. Choose *All Packages* to create differential pairs in all packages.
  - b. Choose *Current Package* to create differential pairs in the selected package.

The differential pair properties are added to all pins that are not part of existing differential pairs based on the naming schemes specified through the `DiffPair_Recognition_Rules` directive.

## Specifying a Naming Convention for Autocreation of Differential Pairs

The `DiffPair_Recognition_Rules` directive in `cds.cpm`, located at `<your_inst_dir>/share/cdssetup/projmgr`, specifies a set of prefixes and suffixes that Part Developer uses to identify differential pairs when you choose the *Auto Create Differential Pairs* menu item from the shortcut (RMB) menu.

The default `DiffPair_Recognition_Rules` definition is as follows:

```
DiffPair_Recognition_Rules 'n:SUFFIX,p:SUFFIX;-
                           -:SUFFIX,+:SUFFIX;_L:SUFFIX,_H:SUFFIX;_LOW:SUFFIX,_HIGH:SUFFIX'
```

The following table shows with an example, pin Y, how each rule in the default `DiffPair_Recognition_Rules` definition impacts the naming of the positive and negative pins of a differential pair:

Differential Pair Recognition Rule	Negative Pin	Positive Pin
<code>n:SUFFIX,p:SUFFIX</code>	YN	YP
<code>-:SUFFIX,+:SUFFIX</code>	Y-	Y+
<code>_L:SUFFIX,_H:SUFFIX</code>	Y_L	Y_H
<code>_LOW:SUFFIX,_HIGH:SUFFIX</code>	Y_LOW	Y_HIGH

You can modify existing rules and add new rules by configuring the `DiffPair_Recognition_Rules` directive in your site CPM file or project CPM file.

**Note:** The negative pin identifier and its affix must be specified first in each differential pair recognition rule.

## Part Developer User Guide

### Creating Parts

---

**Note:** The positive pin and negative pin identifiers in a differential pair recognition rule must have the same affix. In other words, a naming rule such as `n:SUFFIX, p:PREFIX` is not valid.

The following table lists some conventions commonly used by different companies to name differential pair pins:

Negative pins/signals are specified using	Positive pins/signals are specified using	As
<code>_L</code>	<code>_H</code>	Suffix
<code>_LOW</code>	<code>_HIGH</code>	Suffix
<code>_NEG</code>	<code>_POS</code>	Suffix
<code>_N</code>	<code>_P</code>	Suffix
-	<code>+</code>	Suffix
-	<code>+</code>	Prefix

## Naming Differential Pairs

Part Developer uses the `Default_Diffpair_Value` definition in `cds.cpm` to name differential pairs. By default, the `Default_Diffpair_Value` definition is the following:

```
Default_Diffpair_Value 'DP_:Prefix'
```

The value specified in the default definition implies that Part Developer will create a differential pair name by prefixing `DP_` to the basename of the constituent differential pair pins. For example, if two pins `A+` and `A-` constitute a differential pair, the differential pair will be named `DP_A`.

The following table lists some examples of differential pair names as derived by Part Developer.

Default_Diffpair_Value Directive	Pin Names	Pin Names Follow Differential Pair Recognition Rules	Differential Pair Name
<code>Default_Diffpair_Value 'DP_:Prefix'</code>	<code>A+</code> <code>A-</code>	Yes	<code>DP_A</code>

Default_Diffpair_Value Directive	Pin Names	Pin Names Follow Differential Pair Recognition Rules	Differential Pair Name
Default_Diffpair_Value e.g. '_DP:Suffix'	_HD  _LD	Yes	D_DP

You can configure the `Default_Diffpair_Value` definition in your local or site CPM file.

For information on the characters supported in differential pair names, see [Supported Characters for Differential Pair Names](#) on page 564.

## Creating a Differential Pair from Selected Pins

To create a differential pair manually, do the following:

1. Select the constituent pins in the *Logical Pins* grid.
2. Right-click on the selection and choose *Create Differential Pair*.
  - a. Choose *All Packages* to create differential pairs in all packages.
  - b. Choose *Current Package* to create a differential pair in the selected package.

The Differential Pair Name dialog box appears. The *Name* field displays a name for the differential pair that Part Developer derives by adding the prefix or suffix specified in the `Default_Diffpair_Value` directive in `cds.cpm` to the basenames of the pin.

For more information, see [Naming Differential Pairs](#) on page 131.

3. Modify the displayed differential pair name if required.
4. Click *OK* to save the modified differential pairname.

## Deleting Differential Pair Properties

To remove differential pair properties, do the following:

1. Right-click on one of the pins that forms part of the differential pair from which differential pair properties are to be deleted.
2. From the shortcut menu, choose *Remove Differential Pair*.

- a. Choose *All Packages* to remove differential pair properties from pairs in all packages.
- b. Choose *Current Package* to remove differential pair properties from pairs in the selected package.

The differential pair properties are deleted from both of the pins of the differential pair.

**Note:** If you delete one of the pins of a differential pair, the differential pair property is automatically removed from the other pin. For example, if you delete a pin that is the positive pin of a differential pair, the differential pair property from the negative pin of the differential pair is removed. The pin is not deleted.

## Some Quick Pin Mapping Techniques

Part Developer provides the following methods using which the logical/global pins to physical pin mappings can be speeded up:

- Filter Rows
- Hide Mapped Pins
- Global Pin Map – Edit

### Filter Rows

See [Creating a Filter](#) on page 174 for more details.

### Hide Mapped Pins

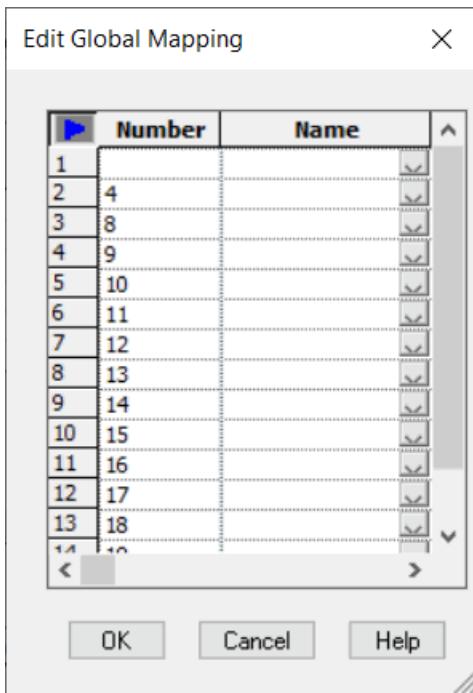
The *Hide Mapped Pins* option enables you to view only the unmapped pins, thus speeding up the mapping process. To hide mapped cols:

- Right-click on a logical/physical/global pin and select *Hide Mapped Pins*.

The pins that are mapped are hidden from the view. For large pin-count parts, this ensures that those pins that are already mapped are hidden from the view and you can concentrate on the unmapped pins.

### Global Pin Map – Edit

This brings up the Edit Global Pin Mapping dialog box.



The *Number* column shows the physical pins that are either mapped to the global pins or are still unmapped. The *Name* column shows the name of the global pin to which a physical pin is mapped. The *Name* column appears empty for the physical pins that are still unmapped. To map multiple physical pins to one global pin, do the following:

1. Select the cells in the *Name* column next to the physical pins to which you want the mapping to be done. For example, if VCC is to be mapped to pins 9,10,11,12, and 13 then multi-select the cells in the *Name* column next to these physical pins.
2. Right-click on the selection and select *Modify Values*.

The Modify Column Values dialog box appears.

3. Select the global pin from the *Name* drop-down list and click *OK*.

The selected global pin gets mapped to the physical pins

## Creating Symbols

Similar to creating packages, you can create symbols in one of the following ways:

- By using the Symbol Editor. See [Using the Symbol Editor](#) on page 135
- Directly from a package. See [Direct Generation from a Package](#) on page 135.

## Using the Symbol Editor

To create packages using the Symbol Editor:

1. Right-click on the *Symbols* entry in the Cell Editor and select *New*.

## Direct Generation from a Package

To directly generate a symbol from a package, do the following:

1. Right-click on a package in the Cell Editor and select *Generate Symbol(s)*.

The Generate Symbols for Package dialog box appears.

2. Select the functions for which you want symbols and click *OK*.

One symbol is created for each function.

In both of the above-mentioned ways, the Symbol Editor appears in the right pane of the Cell Editor.

**Note:** The vector pins will always appear in the expanded form in the symbol. If you want to make the vector bits appear in the collapsed form, use the Symbol Editor to create the symbols. Alternatively, you can select the vector pin bits in the *Logical Pins* grid, right-click on the selection and select *Collapse*.

## Entering Symbol Information

The information for a symbol is provided through the *General* and *Symbol Pins* pages. The Symbol Editor appears with the General page selected by default.

The following information is specified through the General page:

- Symbol properties and their attributes
- Symbol text and their attributes

The *Symbol Pins* page enables you to enter and manage the symbol pin information. For more information about the fields of these two pages, see [Symbol Editor](#) on page 57.

You can begin with either the *General* page or the *Symbol Pins* page. In this guide, information is specified on the *General* page followed by the *Symbol Pins* page. On the General page, do the following:

1. Specify the value for the *PACK\_TYPE* property and determine its attributes.

2. Specify any other symbol properties. To do so, right-click in the *Properties* grid and select *Insert Row After*.
- A blank row appears in the *Properties* grid.
3. Enter the property name, value, and its other attributes.
  4. Specify the text that needs to appear on the symbol. To do so, right-click in the *Text* grid and select *Insert Row After*.

- A blank row appears in the *Text* grid.
5. Enter the symbol text and its attributes.

- Next, you need to specify the pin information.
6. Click *Symbol Pins*.

If you generated the symbol directly from the package, then all the package pins will appear in the *Logical Pins* grid of the symbol. Otherwise, you will need to add pins to the symbol.

7. To add pins to the symbol, click *Pins – Add*.
- The Add Pin dialog box appears.
8. Specify the pins to be added to the symbol and click *OK*.
  9. Save the part.

This completes the process of symbol creation.

## Adding Symbol Pin Properties

To add symbol pin properties, do the following:

1. Select the Symbol Pins page.
  2. Choose *Properties – Add*.
- The Add Properties dialog box appears.
3. Specify the name of the property in the *Name* column and its value in the *Value* column and click *OK*.

**Note:** By default, the properties are added to all the pins of the symbol. To disassociate a property from a particular pin, delete the property value. A null value implies that the property is absent from the pin.

## **Adding Symbol Pin Properties for Individual Bits of a Vector Pin**

To add symbol pin properties to bits of a vector pin, do the following:

1. Select the Symbol Pins page.
2. Right-click on the vector pin and select *Expand*.
3. Choose *Properties – Add*.

The Add Properties dialog box appears.

4. Specify the name of the property in the *Name* column and its value in the *Value* column and click *OK*.

**Note:** By default, the properties are added to all the pins of the symbol. To disassociate a property from a particular pin, delete the property value. A null value implies the property is absent from the pin.



### **Caution**

***Collapsing the bits of a vector pin may result in the loss of bit-specific pin properties.***

## **Adding Images to Symbols**

To add an image to a symbol, do the following:

1. Open the symbol in Symbol Editor.
2. Choose *Graphic Editor – Draw – Image*.  
The Open dialog box appears.
3. Select the image file and click *Open*.
4. Move the mouse pointer to the Symbol Viewer and click on the location where you want the image to appear.
5. Press Esc to complete the process of adding the image to the symbol.
6. Repeat steps 2 to 5 for each additional image to be added to the symbol.

## Adding Pins in Additional Packages and Symbols

Whenever a package or a symbol is created, the pins that were entered for it are available for subsequent packages and symbols as well. By default, they will appear unselected in the pins grid of the Add Pin dialog box. To add them to the packages or symbols, simply select the pins and click *OK*.

## Creating Sizeable and HAS\_FIXED\_SIZE Symbols

To create sizeable and `HAS_FIXED_SIZE` symbols, do the following:

1. Create a symbol, `sym_1`, through the Symbol Editor and add the sizeable pins.
2. Create a package.
3. Create the necessary slots in the package.
4. Copy `sym_1` to `sym_2`.
5. For `sym_2`, specify the value of the `SIZE` property on the basis of the number of slots in `sym_1`.

This will add the `HAS_FIXED_SIZE` property to `sym_2`.

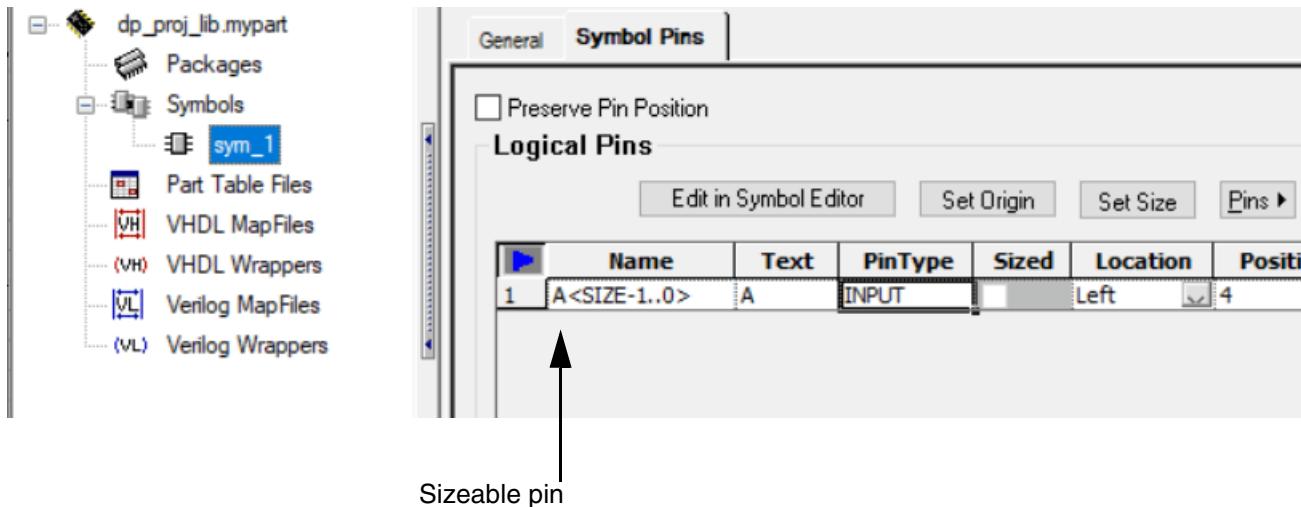
Detailed steps follow with an example of a four-slot part with one input pin:

1. Create a new symbol using Symbol Editor.
2. On the Symbol Pins page of the Symbol Editor, choose *Pins – Add*.  
The Add Pin dialog box appears.
3. Select the *Sizeable* option and enter the pin name.
4. Specify the `<SIZE-1..0>` or `<SIZE..1>` notation for creating the sizeable pin.
5. Specify the pin-related information, such as pin type, location, checks, and load values and click on *Add*.
6. To add the pin to the symbol, click *OK*.

## Part Developer User Guide

### Creating Parts

The sizeable pin is added to the symbol.



Next, create a package from the symbol.

7. Right-click on *sym\_1* and choose *Generate Package*.

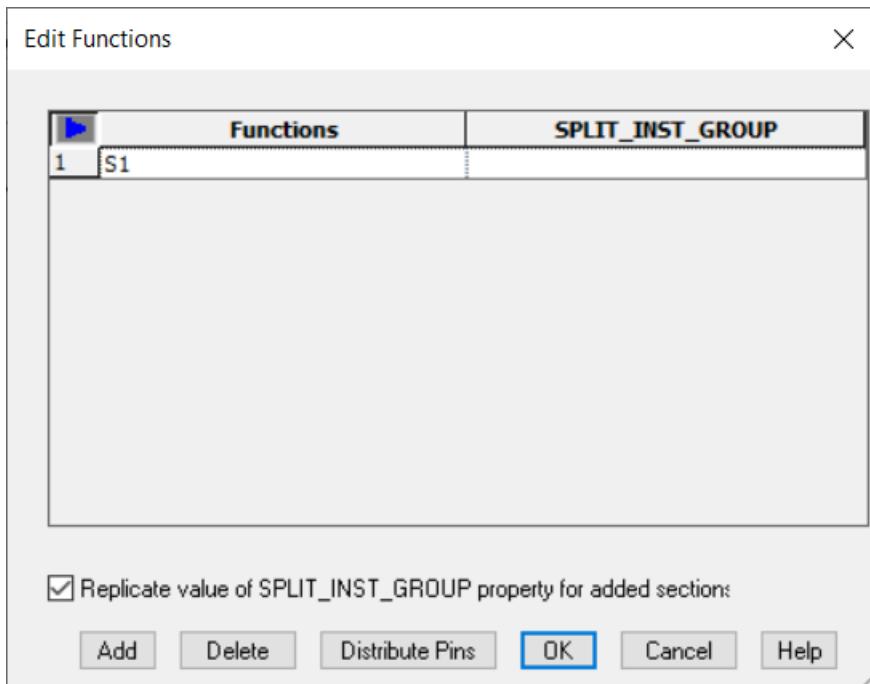
The package is generated for the symbol. By default, only one slot is created. You will need to add slots as required.

8. To create slots, click *Functions/Slots* on the Package Pin page of the Package Editor.

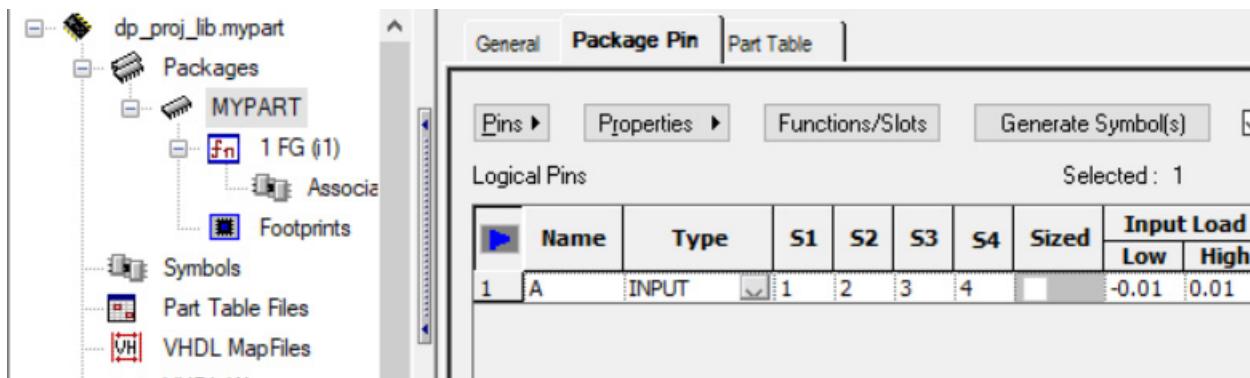
## Part Developer User Guide

### Creating Parts

The Edit Functions dialog box appears.



9. Click **Add** to increase the number of slots.
10. Click **OK**.
11. Specify pin numbers for the four slots. For example, specify pin numbers 1, 2, 3, and 4 to be mapped to the four slots for pin A.

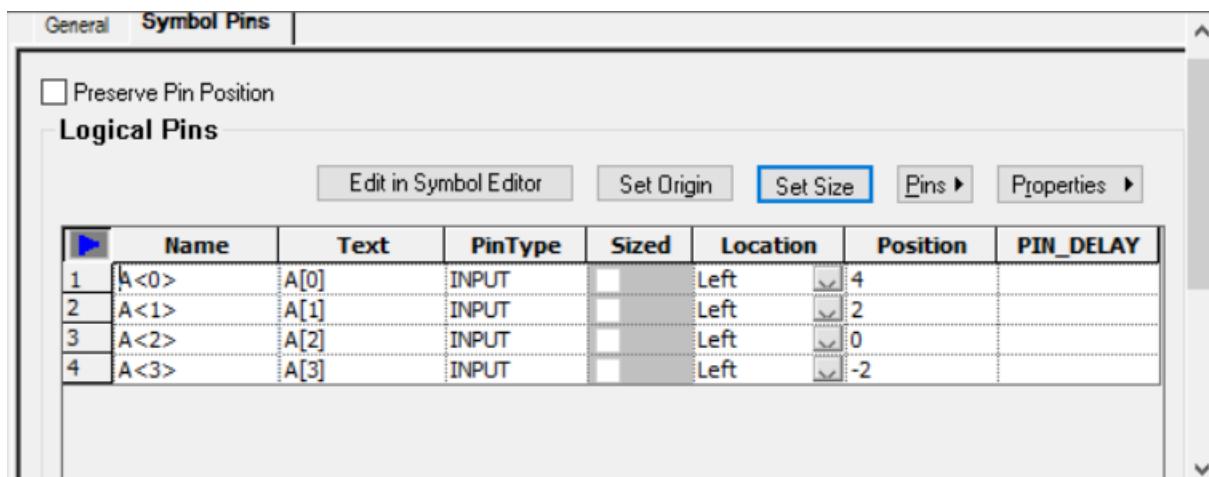


Next, create sym\_2 by making a copy of sym\_1.

12. Open sym\_2 in Symbol Editor by clicking on sym\_2.
13. Click **Set Size**.

The Specify The Symbol Size dialog box appears.

14. Enter the value of the size property in the *New Size* field. The value will depend on the number of slots in which the pin is present. For example, for pin A, Size will be replaced by 4, since it is present in 4 slots.



15. Save the part.

The `HAS_FIXED_SIZE` property will be automatically added to the `symbol.css` file of `sym_2`. This property is not visible from within the Symbol Editor.



***Do not change the pin\_name <0> pin for HAS\_FIXED\_SIZE symbols. This may lead to deletion of all the bits of the pin. For example, renaming A<0> may lead to deletion of all the bits of the pin A.***

## Creating Parts with Bubble Group and Pass-Through Pins

### Bubble Group and Pass-Through Pins

`Bubble_Group` is a property that can be added to two or more pins to control their assertion state. You can create bubble groups in Part Developer by using the pin attributes. Pass-through pins can be created by duplicating the pins and manipulating their pins shapes.

### Creating Bubble Groups

1. Open the Symbol Editor and select the pins that need to be bubble-grouped.

**Note:** Do not select the entire row. Selecting the entire row prevents the changing of pin

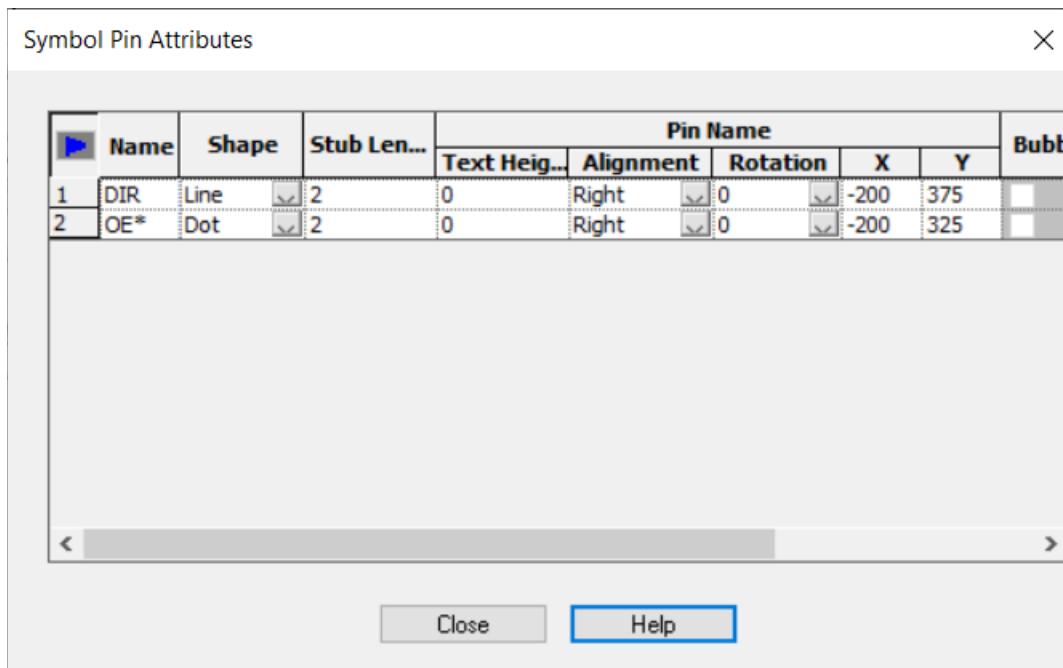
## Part Developer User Guide

### Creating Parts

attributes.

#### 2. Click *Pins – Attributes*.

The Symbol Pin Attributes dialog box appears.



#### 3. Change the shape of the pins to *Bubble* and click *Close*.

The pin shape changes to bubble.



Next, you need to add the *BUBBLE\_GROUP* property to the pins.

#### 4. Click *Properties – Add*.

The Add Properties dialog box appears.

#### 5. Enter *BUBBLE\_GROUP* in the *Name* field and click *OK*.

#### 6. Specify the value for the *BUBBLE\_GROUP* property as required.

## Creating Pass-Through Pins

To view the information on default Symbol Editor, [Symbol Editor User Guide](#).

1. Open Symbol Editor and select the pins that need to be made pass-through.
2. Make a copy of the pins and select both the original and copied pins.

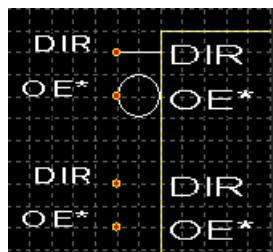
**Note:** Do not select the entire rows. Selecting entire rows prevents the changing of pin attributes.

3. Click *Pins – Attributes*.

The Symbol Pin Attributes dialog box appears.

4. Change the pin shapes of the copied pins to *Zero* and click *Close*.

The pin shapes change to zero and are placed two grids away from the symbol outline.



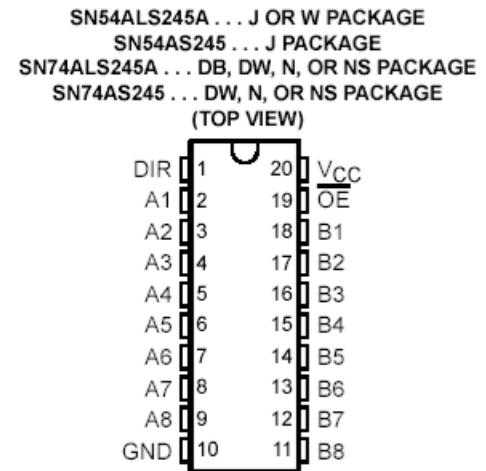
5. Move the pins as required.

For example, consider the sn74als245a as shown below:

## Part Developer User Guide

### Creating Parts

---



FUNCTION TABLE

INPUTS		OPERATION
$\overline{OE}$	DIR	
L	L	B data to A bus
L	H	A data to B bus
H	X	Isolation

If you follow the above-mentioned steps to create the Bubble\_Group and pass-through pins for pins DIR and OE\*, you will get two DIR and OE\* pins, with one set of DIR and OE\* with shape as bubble and with the BUBBLE\_GROUP property and the other set with shape as zero.

## **Creating Parts from PDFs**

---

Part Developer provides the ability to create parts from datasheets available in the PDF format. You can copy the pin information text from a datasheet and paste the information directly into *Logical* and *Physical Pins* grids. It is also possible to create parts from the pin information stored in grids and tables.

After specifying the pin information, you can complete the part creation steps detailed in [Chapter 5, “Creating Parts.”](#)

The steps to create a part from a PDF are explained through the use of a Pentium datasheet.

### **Pentium Datasheet**

The following illustration is a relevant portion from the datasheet of the Pentium processor. Note that only a partial pin list is displayed.

**Table 30. Pin Listing by Pin Name**

Pin Name	Pin Number	Signal Buffer Type	Direction
A3#	F30	Source Synch	Input/Output
A4#	C29	Source Synch	Input/Output
A5#	D30	Source Synch	Input/Output
A6#	C31	Source Synch	Input/Output
A7#	F28	Source Synch	Input/Output
A8#	D28	Source Synch	Input/Output
A9#	F26	Source Synch	Input/Output
A10#	C23	Source Synch	Input/Output
A11#	A31	Source Synch	Input/Output
A12#	C25	Source Synch	Input/Output
A13#	A25	Source Synch	Input/Output
A14#	A23	Source Synch	Input/Output
A15#	A27	Source Synch	Input/Output

These are the following ways in which you can enter pin information into Part Developer:

- Directly into Part Developer

## Part Developer User Guide

### Creating Parts from PDFs

This method requires you to individually copy the pin name and pin number columns in Part Developer and then manually update the pin type information.

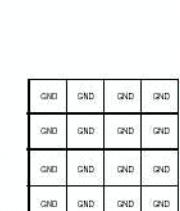
#### ■ Copying into Excel/Star Office and then copying the information into Part Developer

This method has the benefit of copying the entire pin name/type and mapping information into Part Developer in a single step. The pin type information is also updated automatically.

#### ■ Importing Pin Grid

This method provides you the ability to import pin information stored in grid format in a PDF file. In the grid format, the combination of row name and column name is used to get the pin number. For example, in the following figure, the combination of row number 1 and column name Y is used to get the pin number Y1, which is named NC.

Y	W	V	U	T	R	P	N	M	L	K	J	H	
1	NC	GND	EOT	NC	NC	V <sub>I01</sub>	FUL1	FUL4	FULH	BH00	V <sub>I02</sub>	BH01	LH01
2	NC	NC	ACK	FULL	NC	FUD1	NC	FUL6	FUL2	BH01	BH12	V <sub>I03</sub>	LH05
3	DQ64	NC	NC	V <sub>I03</sub>	V <sub>I03</sub>	V <sub>I03</sub>	NC	NC	V <sub>I03</sub>	BH02	V <sub>I03</sub>	LH01	LH14
4	DQ62	NC	V <sub>I03</sub>	GND	RSTL	NC	FUL00	GND	FUL5	FUL6	BH11	LH00	GND
5	DQ80	V <sub>I02</sub>	NC	DQ86									
6	V <sub>I31</sub>	NC	DQ86	DQ88									
7	DQ69	V <sub>I032</sub>	DQ62	DQ64									
8	NC	DQ48	DQ48	GND									
9	DQ40	DQ42	V <sub>I032</sub>	DQ44									
10	V <sub>I03</sub>	NC	DQ29	DQ38									
11	V <sub>I032</sub>	DQ84	DQ32	DQ30									
12	NC	DQ28	V <sub>I030</sub>	DQ26									
13	DQ24	V <sub>I02</sub>	DQ20	GND									

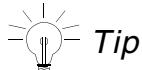


#### ■ Importing Pin Table

This method provides you the ability to import pin information stored in table format in a PDF file. In the table format, the pin information is stored in two or more columns per row. The columns typically describe the pin name, number, type and so on. For example, the following table describes the pin name, physical pin number, buffer type, and pin type in table format.

**Table 30.** Pin Listing by Pin Name

Pin Name	Pin Number	Signal Buffer Type	Direction
A3#	F30	Source Synch	Input/Output
A4#	C29	Source Synch	Input/Output
A5#	D30	Source Synch	Input/Output
A6#	C31	Source Synch	Input/Output
A7#	F28	Source Synch	Input/Output
A8#	D28	Source Synch	Input/Output
A9#	F26	Source Synch	Input/Output
A10#	C23	Source Synch	Input/Output
A11#	A31	Source Synch	Input/Output
A12#	C25	Source Synch	Input/Output



*Tip*

You can use Acrobat Reader 5.1 and above to read the PDF files. These versions have a *Column Select* option, which ensures that the pin names that are copied from the PDF file are pasted as individual pins in the pin grid. Copying pin information from earlier versions of Acrobat Reader results in all the pin names appearing as a single pin name in the *Logical Pins* grid. To fix this, use the *Edit – Paste Special(Grid) – Convert Whitespaces to NewLine* option when pasting data into Part Developer.

## Directly into Part Developer

1. Open the datasheet in Acrobat Reader.
2. Select the *Pin Name* column. Press Ctrl and drag cursor to select the complete column. Ctrl+C to copy the column.
3. Launch Part Developer and, if required, create a new part and package.
4. Go to the Package Pin page of the Package Editor.
5. Press Ctrl + I to insert a new row in the *Logical Pins* grid.
6. Select the empty cell under the *Name* column and press Ctrl + V to paste the pin names into the *Logical Pins* grid.

# Part Developer User Guide

## Creating Parts from PDFs

---

The pin names appear under the *Name* column.

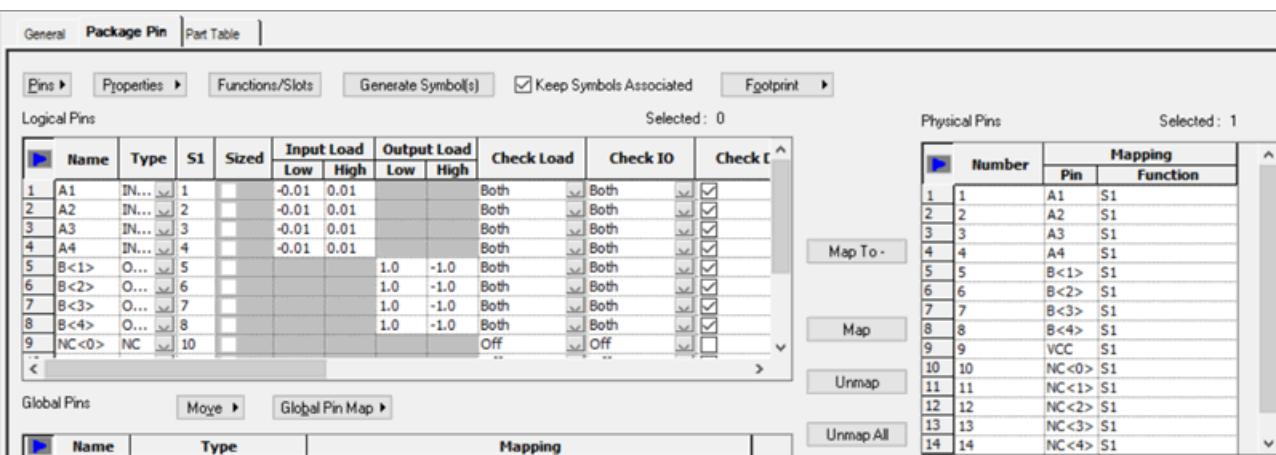
Logical Pins			
	Name	Type	S1
1	A1	IN...	1
2	A2	IN...	2
3	A3	IN...	3
4	A4	IN...	4
5	B<1>	O...	5
6	B<2>	O...	6
7	B<3>	O...	7
8	B<4>	O...	8
9	NC<0>	NC	10
10	NC<1>	NC	11
11	NC<2>	NC	12
12	NC<3>	NC	13
13	NC<4>	NC	14
14	VCC	PO...	9

**Note:** All the pins are of the type UNSPEC. You will need to manually update the pin types for all the pins.

Next, you need to copy the pin numbers that are mapped to the pin names.

7. Select the *Pin Number* column in the datasheet and press Ctrl + C.
8. Select the first cell under the *S1* column and press Ctrl + V.

The physical pin numbers are copied into the *Logical Pins* grid. The *Physical Pins* grid is also updated automatically with the pin mapping information.



The screenshot shows the Part Developer software interface with the following components:

- Top Bar:** General, Package Pin, Part Table.
- Tool Buttons:** Pins ▶, Properties ▶, Functions/Slots, Generate Symbol(s), Keep Symbols Associated, Footprint ▶.
- Logical Pins Grid:** A table with columns: Name, Type, S1, Sized, Input Load (Low, High), Output Load (Low, High), Check Load, Check IO, Check I<sub>C</sub>. It lists pins 1 through 14 with their respective names and types.
- Physical Pins Grid:** A table with columns: Number, Pin, Function. It lists pins 1 through 14 with their corresponding physical pin numbers and functions.
- Buttons:** Map To..., Map, Unmap, Unmap All.
- Global Pins:** Global Pins, Move ▶, Global Pin Map ▶.
- Bottom Grid:** A table with columns: Name, Type, Mapping.

**Copying into Excel/Star Office and then copying the information into Part Developer**

1. Open the datasheet in Acrobat Reader.
2. Select the *Pin Name* column. Press Ctrl and drag cursor to select the complete column. Ctrl+C to copy the column.



When pasting pin locations from the spreadsheet, ensure that the values of the pin location names follow the correct syntax, that is, that the first letters of Left, Bottom, Right, and Top are in uppercase and the rest of the letters are in lowercase.

3. Open an Excel spreadsheet and press Ctrl + V to paste the data in the first column.
4. Next, copy the *Direction* and *Pin Number* columns and paste into the columns adjacent to the pin name column.

The filled spreadsheet should appear like the following illustration:

	A	B	C
1	COMP0	Input/Output	AU27
2	COMP1	Input/Output	F24
3	D0#	Input/Output	Y38
4	D1#	Input/Output	AD36
5	D2#	Input/Output	W37
6	D3#	Input/Output	AE37
7	D4#	Input/Output	AG39
8	D5#	Input/Output	AA35

5. Select the three columns in the spreadsheet and press Ctrl + C.
6. Press Ctrl + I to insert a blank row in the *Logical Pins* grid.
7. Select the empty cell under the *Name* column in *Logical Pins* grid and press Ctrl + V.

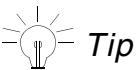
The pin information along with the pin type and the mapping information is copied into the *Logical Pins* grid. The *Physical Pins* grid is also updated automatically. Note that the Input/Output pin type is automatically converted to the BIDIR type in the

*Logical Pins* grid. This translation is controlled through the `translate.cpm` file located in the `<your_inst_dir>\share\cdssetup\LMAN` location.

Name	Type	S1	Sized	Input Load		Output Load		Check Load	Check IO
				Low	High	Low	High		
1 COMPO	BIDIR	AU27		-0.01	0.01	1.0	-1.0	Both	Both
2 COMP1	BIDIR	F24		-0.01	0.01	1.0	-1.0	Both	Both
3 D0#	BIDIR	Y38		-0.01	0.01	1.0	-1.0	Both	Both
4 D1#	BIDIR	AD36		-0.01	0.01	1.0	-1.0	Both	Both
5 D2#	BIDIR	W37		-0.01	0.01	1.0	-1.0	Both	Both
6 D3#	BIDIR	AE37		-0.01	0.01	1.0	-1.0	Both	Both



**When copying data from PDFs, invalid characters in pin names need to be fixed manually. Part Developer will generate errors if an attempt is made to save a part with invalid characters in pin names.**



Sometimes, a PDF might have multiple pins with the same name. On copying, such pin names will be displayed as errors. For example, a part may have multiple pins with the name GND. On copying, these pin names will appear in red indicating an error. You can convert such duplicate pin names to the bits of a vector pin. For more information, see [Converting Scalar Pins to Vector](#) on page 167.

## Importing Pin Grid

### Steps

1. Choose *File – Import and Export*.  
The Import and Export wizard appears.
2. Choose *Import Pin Grid* and click *Next*.  
The Select Destination page appears.
3. Specify the cell name and select the library in which it is to be created and click *Next*.  
The Paste the data page appears.

4. Open the PDF in Adobe Acrobat, select the complete table, right click to copy the table with formatting.
5. Right-click in the Paste the data page and select the *Paste* option.

The data is pasted into the Paste the data page. Note that the pasted data may need some editing before it can be used to create the part. You can do several such editing tasks by right-clicking on the pasted data and selecting the appropriate option. For example, the pin names may be split into two or more rows. For example, a pin named Data45 may appear as Data4 in the first row and 5 in the second row. By selecting the *Merge Rows* option, 5 is appended to Data4 and the pin name is changed to Data45. Similarly, you can remove empty rows by selecting the *Delete Empty Rows* option. For more information on available options, see [Right-Click Options on Paste the Data Page](#) on page 152.

6. Click *Next*.

The Preview the Derived Data page appears. You can view the part information and make any changes if necessary.

7. Click *Finish* to complete part creation.

If there are duplicate pins in the imported data, the Duplicate Pin Resolver dialog appears. You need to resolve the duplicate pins. For more information, see [Duplicate Pin Name Handling](#) on page 251.

Finally, the Cell Editor appears with the part information.

## Importing Pin Table

### Steps

1. Choose *File – Import and Export*.  
The Import and Export wizard appears.
2. Select *Import Pin Table* and click *Next*.  
The Select Destination page appears.
3. Specify the cell name and select the library in which it is to be created and click *Next*.  
The Paste the data page appears.
4. Open the PDF in Adobe Acrobat, select the complete table, right click to copy the table with formatting.

**5.** Right-click on the Paste the data page and select the *Paste* option.

The data is pasted into the Paste the data page. Note that the pasted data may need some editing before it can be used to create the part. You can do several such editing tasks by right-clicking on the pasted data and selecting the appropriate option. For example, most tables may have column headings. Since this contains information, you can right-click on the row that contains the columns headings and select the *Remove Selected Row* option to delete it. Similarly, the pin names may appear in more than one column. Likewise, other pin information can also span multiple columns. To ensure that all pin names are read correctly, right-click on the column that has pin names and select the *Set Cols as Pin Name* option. This will change the color of the column. Now, right-click on the next column that contains the pin number and select the *Set Cols as Pin Name* option again. Note that the color of the selected column changes. Both the columns that have been selected to be pin names will have the same color. Similarly, select the columns that have pin type information as choose the *Set Cols as Pin Type* option. You may want to remove the other columns that contain information that is not required in the `chips.prt` file. For more information on available options, see [Right-Click Options on Paste the Data Page](#) on page 152.

**6.** Click *Next*.

The Preview the Derived Data page appears. You can view the part information and make any changes if necessary.

**7.** Click *Finish* to complete part creation.

**Note:** Various pin types are automatically converted to supported pin types by using the definitions provided in the `PinTable` section of the `propfile.prop` file. In addition, pin types are also derived from pin names specified in the `PinTablePinName` section of the `propfile.prop` file.

## Right-Click Options on Paste the Data Page

---

Menu Option	Description
Set Col as Pin Number	Sets the selected column as the source for pin numbers.
Set Col as Pin Name	Sets the selected column as the source for the pin name.
Set Col as Pin Type	Sets the selected columns as the source for the pin types.

## Part Developer User Guide

### Creating Parts from PDFs

---

Menu Option	Description
Shift Up	Moves up the contents of the selected cells by one row.   <b><i>Existing data in the cell into which the contents of the selected cell are moved is deleted permanently.</i></b>
Shift Down	Moves down the contents of the selected cells by one row.   <b><i>Existing data in the cell into which the contents of the selected cell are moved is deleted permanently.</i></b>
Shift Left	Moves to the left the contents of the selected cells by one column.   <b><i>Existing data in the cell into which the contents of the selected cell are moved is deleted permanently.</i></b>
Shift Right	Moves to the right the contents of the selected cells by one column.   <b><i>Existing data in the cell into which contents of the selected cell is moved is deleted permanently.</i></b>
Shift..	Moves the contents of the selected cells by a specified number of rows/columns and direction.   <b><i>Existing data in the cell into which contents of the selected cell is moved is deleted permanently.</i></b>
Delete Column	Deletes the selected column.

## Part Developer User Guide

### Creating Parts from PDFs

---

Menu Option	Description
Merge Selected Cells	Merges the data of the selected cells into the anchor cell. This option is valid only when importing data from pin grids. The merging is possible only when the data is split into multiple rows under a specific row number.
Merge	Automatically merges the data in the complete grid. This option is valid only when importing data from pin grids. The merging is possible only when the data is split into multiple rows under a specific row identifier.
Paste	Pastes the contents of the clipboard into the <i>Paste the data</i> page.
Set Row as Header	Sets the selected row as the header identifier.
Set Col as Header	Sets the selected column as the header identifier.
Merge Selected Portion	Merges the selected portion into the first row of the specific row identifier.

---

---

# Modifying Parts

---

Using Part Developer, you can modify existing parts. You can do the following to modify the parts:

- Modify the logical pin list
- Modify the packages
- Modify the symbols
- Modify the Verilog wrappers/map files
- Modify the VHDL wrappers/map files

**Note:** Any change to package/symbol pins may cause the package-symbol associations to break. You can use Interface Comparator to synchronize the packages with the symbols and vice versa.

When synchronizing packages with symbols, the package pin list may also be modified. To ensure that associated symbols are updated automatically whenever the package pin list is modified, select the *Keep Symbols Associated* option in Package Editor.

## Modifying Logical Pins

In Part Developer, you can modify logical pins in two contexts:

- For specific packages and symbols
- For all packages and symbols

To modify pins for specific packages and symbols, use the Package Editor and Symbol Editor. To modify pins for all packages and symbols, use the *Pins* option.

You can modify logical pins in the following ways:

- Rename pins

- Modify attributes of pins, such as location, type, load values, checks, and values of properties on pins

## Renaming Pins

### Symbol Pin Rename

To rename a symbol pin, do the following:

1. Select the symbol in the Cell Editor.

The symbol information appears in the legacy Symbol Editor.

2. Click *Symbol Pins*.

3. Double-click on the pin name to be renamed.

4. Change the pin name.

5. Choose *File – Save*.

The pin is renamed in the selected symbol.

### Package Pin Rename

To rename a package pin, do the following:

1. Select the package in the Cell Editor.

The package information appears in the Package Editor.

2. Click *Package Pin*.

3. Double-click in the *Name* column of the pin to be renamed.

4. Change the pin name.

5. Choose *File – Save*.

The pin is renamed in the selected package.

### Pin Rename across All Packages and Symbols

To rename a pin across all packages and symbols, do the following:

1. Select a package/symbol in which the pin is present.

## Part Developer User Guide

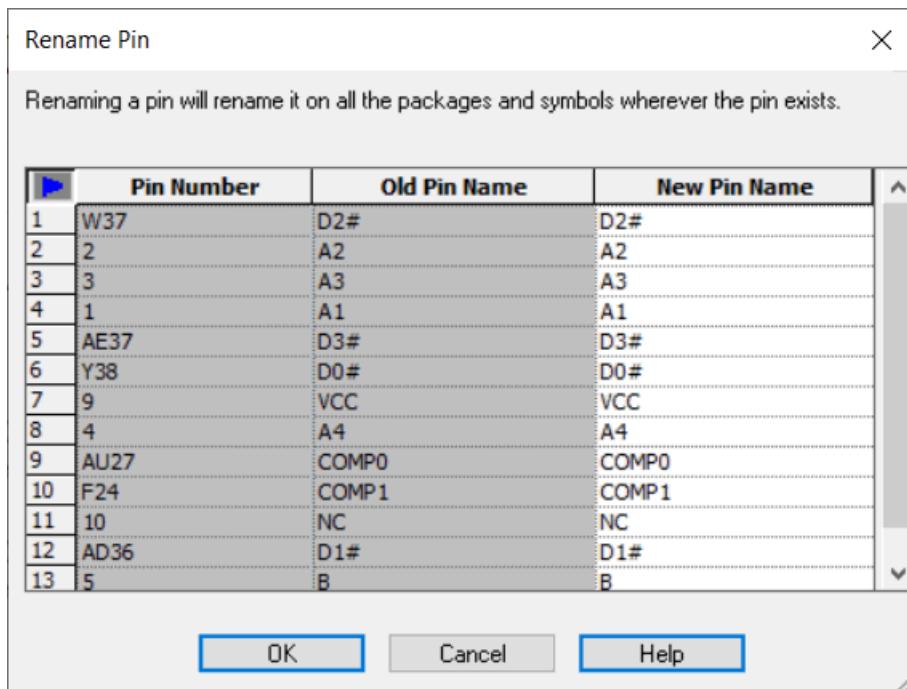
### Modifying Parts

The package/symbol information appears in the Package Editor/Symbol Editor.

2. Click *Package Pin/Symbol Pins*.

3. Choose *Pins – Global Rename*.

The Rename Pin dialog box appears. This dialog box lists all the pins that are on the package/symbol. The Pin Number column displays a comma-separated list of pin numbers corresponding to the logical pin names across all slots and packages.



4. Select the pin that is to be renamed.

5. Enter the new name for the pin and click *OK*.

The pin will be renamed in all the symbols/packages where the pin is present.

## Other Pin Modifications

### Symbol Pin Modification

The following can be modified for a symbol pin:

- Pin name

- Pin text
- Pin location
- Pin position
- Value of properties on the pins

To modify a symbol pin, do the following:

1. Select the symbol in the Cell Editor.

To view the information on default Symbol Editor, [Symbol Editor User Guide](#).

2. Click *Symbol Pins*.

3. Modify the pins as required.

**Note:** The value that you enter in the *Text* column is used as the value of the `PIN_TEXT` property.

**Note:** The *PinType* entries appear disabled since pin type is not applicable for a symbol pin. Also, the *Size* property cannot be modified for a symbol pin.

4. Choose *File – Save*.

The pin is modified in the selected symbol.

## Package Pin Modification

The following can be modified for a symbol pin:

- Pin name
- Pin type
- Pin mapping
- Pin load values
- Pin check options
- Value of properties on the pins

To modify a package pin, do the following:

1. Select the package in the Cell Editor.

The package information appears in the Package Editor.

- 2.** Click *Package Pin*.
- 3.** Select the pin to be modified and make the necessary changes.

**Note:** The *Sized* field appears disabled since it is not applicable for a package pin.

- 4.** Choose *File – Save*.

The pin is modified in the selected package.

### **Pin Modification across All Packages and Symbols**

The following can be modified for a symbol pin:

- Pin type
- Pin location
- Pin loads
- Pin check options

To modify a pin across all packages and symbols, do the following:

- 1.** Select a package/symbol.

The package/symbol information appears in the Package Editor/Symbol Editor.

- 2.** Click *Package Pin/Symbol Pin*.

- 3.** Choose *Pins – Modify*.

The Modify Pin dialog box appears. All the pins that are present on the part are listed.

- 4.** Modify the pins as required and click *OK*.

The pins will be modified in all the symbols/packages.

## **Modifying Packages**

Packages are modified through the Package Editor. You can modify packages by doing the following:

- [Modifying and Deleting Logical and Physical Parts](#)
- [Modifying Package Properties](#)
- [Modifying Footprint Information](#)

- [Modifying Pin Lists and Mapping](#)
- [Modifying Package Pin Properties](#)
- Modify a package by deleting pins

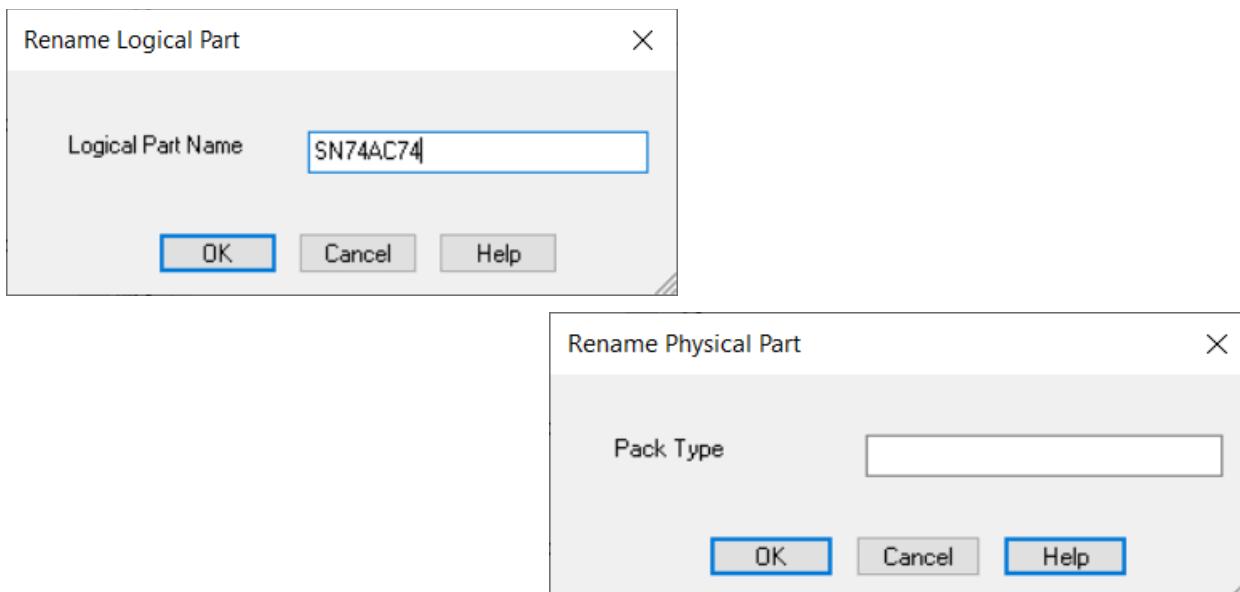
## Modifying and Deleting Logical and Physical Parts

Logical and physical parts can be modified (renamed) and deleted.

1. To rename a logical or a physical part, do the following:

- To rename a logical part, right-click on the logical part name and select *Rename*.
- To rename a physical part, right-click on the physical part name and select *Rename*.

The Rename Logical Part or Rename Physical Part dialog box appears, depending on the part you chose in the first step.



2. Specify the logical or physical part name and click *OK*.

The logical or physical part name is updated. If you renamed a logical part, note that the physical part names are also updated with the modified logical part name, and vice versa.

Renaming a package does not automatically rename the package in the map file. Therefore, an error message prompting you that the package is missing is generated when validation is run for the part.

To delete a logical or a physical part, right-click on the logical or physical part name and select *Delete*. When you delete a logical part, all its associated physical parts are deleted.

## Modifying Package Properties

The following package properties can be modified:

- Class of the package
- Refdes prefix
- Add/remove/change the value of any additional properties on package

Do the following to modify the package properties:

- To change the class of the package, select a new value from the *Class* drop-down list.
- To change the refdes prefix, either select a value from the list or enter a new value.
- To modify an additional property, select the property in the *Additional Properties* grid and change its name/value.
- To add a new property, press *Ctrl + I* in the *Additional Properties* grid and enter the property name and its value.
- To delete a property, right-click on the property row and select *Delete Selected Rows*.
- To delete multiple properties, press *Shift* and click the row number columns. Next, right-click on the selection and select *Delete Selected Rows*.

## Modifying Footprint Information

Modifying footprint information involves modifying the values of the *Jedec\_Type* and *Alt\_Symbols* fields.

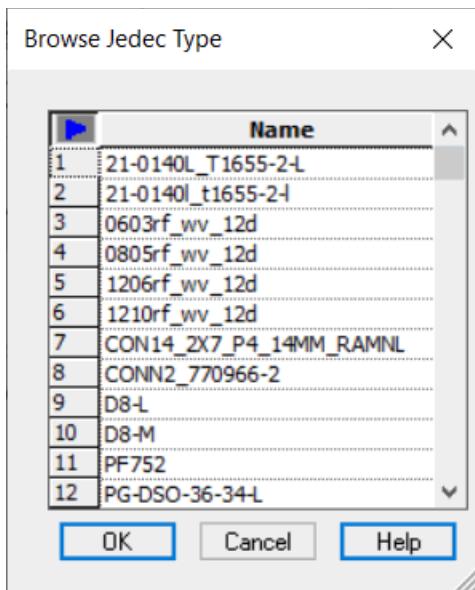
To update the *Jedec\_Type* field:

1. Click the browse button next to the *Jedec\_Type* field.

The Browse Jedec Type dialog box appears.

## Part Developer User Guide

### Modifying Parts

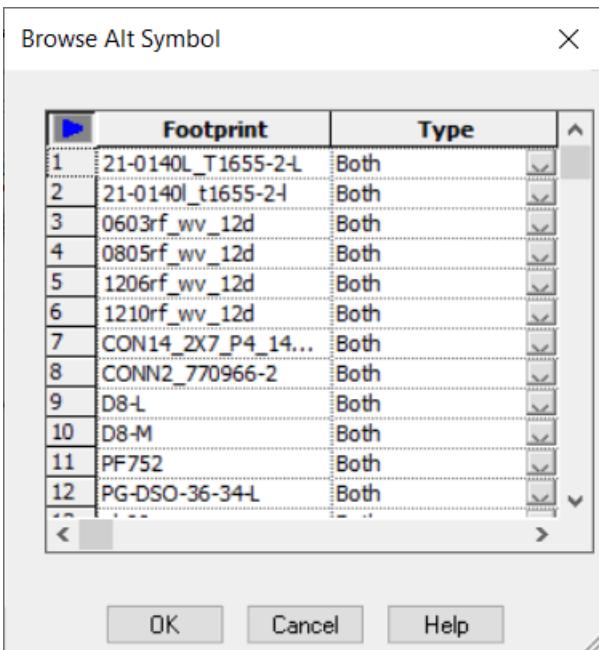


2. Select a footprint from the list of available footprints and click *OK*.

To update the *Alt\_Symbols* field:

1. Click the browse button next to the *Alt\_Symbols* field.

The Browse Alt Symbols dialog box appears.



2. Select footprint(s) and their types and click *OK*.

**Note:** Part Developer uses the value of the PSMPATH variable for determining the path to the footprint files. This variable is set when you install Part Developer. By default, this variable points to the location where the Cadence-supplied Allegro footprints are stored. You can modify the value of PSMPATH through the *PCB Editor Setup* option. To update the PSMPATH:

- a. Choose *Tools – PCB Editor Setup*.

The User Preferences Editor dialog box appears.

- b. Click the *Paths – Library* entry in the *Categories* list.

- c. Click the browse button (the button with three dots) next to the *psmpath* entry under Preference.

The psmpath Items dialog box appears. You can set up your preferences for the location of footprints in this dialog box. You can see the current value of the PSMPATH environment variable by selecting the *Expand* check box.

- d. To add new footprints, click on the *New* button. It is the button with a dotted outline.

A new blank line is added with a browse button.

- e. Browse and select the directory in which the .dra and .psm files are stored and click *OK*.

- f. Click *OK* again to close the User Preferences Editor dialog box.

## Modifying Pin Lists and Mapping

The Package Pin List can be modified by doing the following:

- modifying logical pins. See [Modifying Logical Pins](#) on page 155 for more details.
- modifying physical pins
- changing the number of slots in a package
- moving a pin POWER, NC, or GROUND pin into the pin section or vice-versa
- changing the mapping information between the logical and physical pins

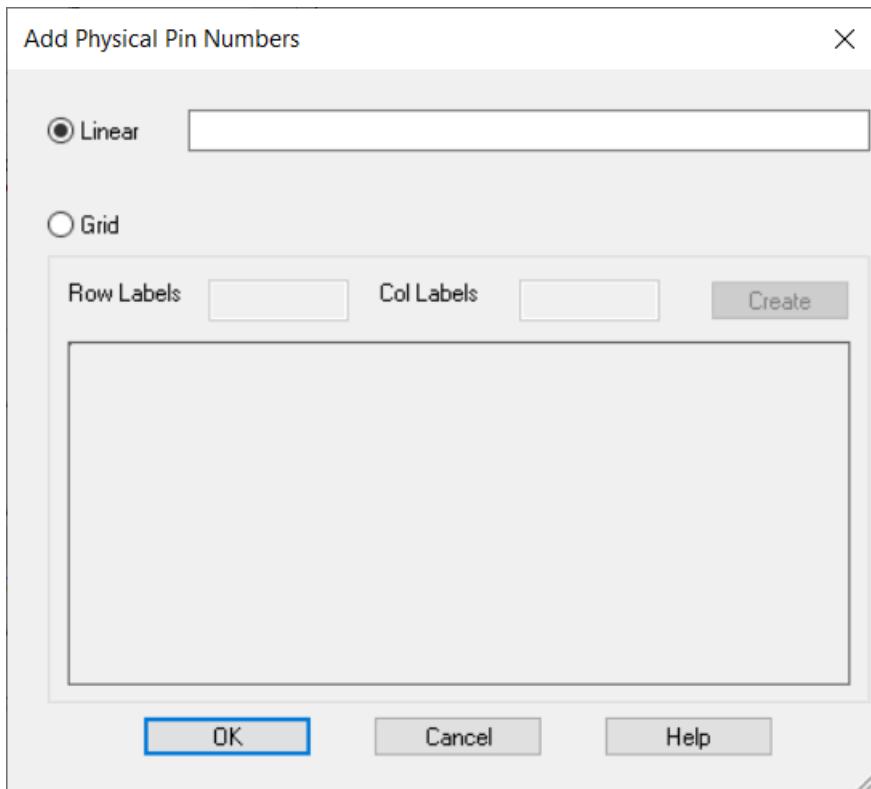
To add physical pins, do the following:

1. To add more physical pins, click *Footprint – Add Physical Pins Manually*.

## Part Developer User Guide

### Modifying Parts

The Add Physical Pin Numbers dialog box appears.



2. Enter the pin numbers in either linear or grid format and click *OK*.

The added physical pins will appear in the *Physical Pins* list.

To delete physical pins:

1. Unmap the physical pins to be deleted.
2. Select the physical pin numbers.
3. Right-click on the selection and select *Delete Selected Rows*.

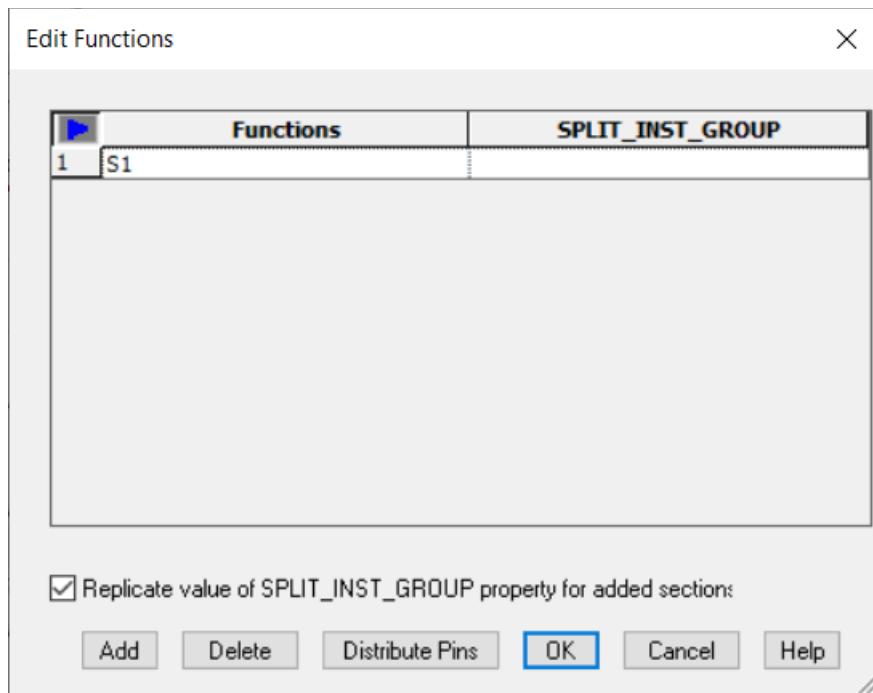
To change the number of slots in a package:

1. Click *Functions/Slots*.

The Edit Functions dialog box appears.

## Part Developer User Guide

### Modifying Parts

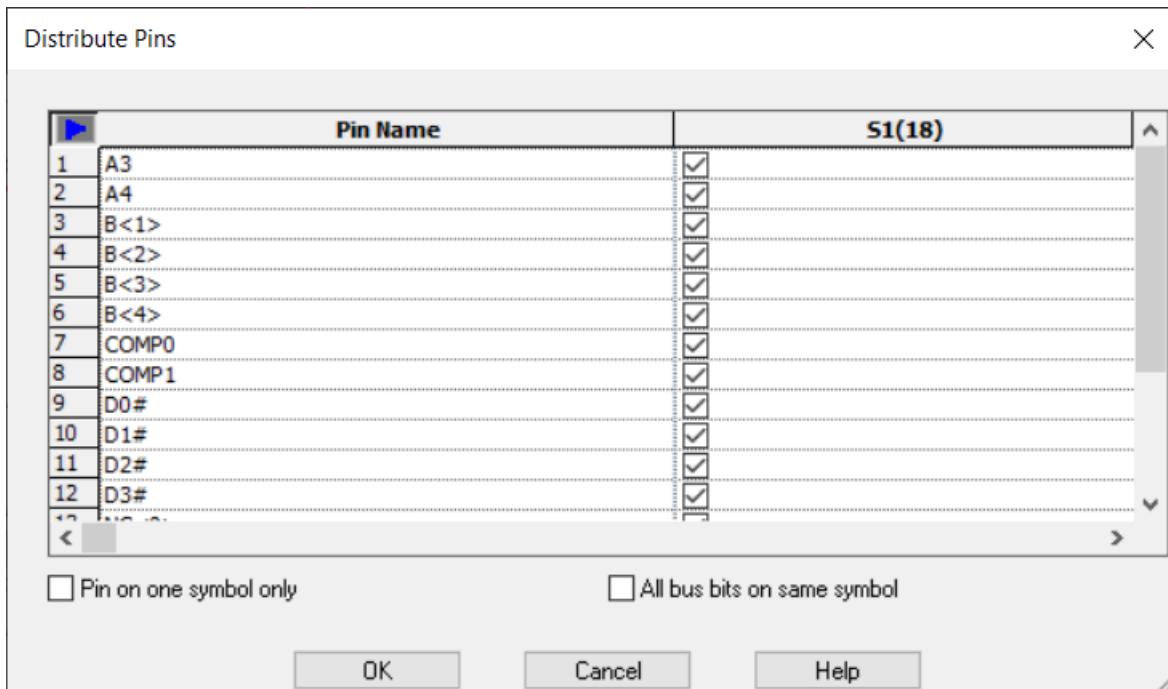


2. To add more slots, click *Add*.
3. To remove slots, click *Remove*.
4. After the slot information has been modified, you should redistribute the pins across slots by clicking *Distribute Pins*.

The Distribute Pins dialog box appears.

## Part Developer User Guide

### Modifying Parts



Distribute the pins across the slots as required. If the part is split, you can ensure that a pin is present in one and only one slot for a split part by selecting *Pin on one symbol only* and *All bus bits on same symbol* check boxes. This ensures that a pin will be present in only one symbol (since each slot will be represented by a unique symbol).

#### 5. Click **OK**.

To move a pin to the *Global Pins* grid:

1. Select the pin in the *Logical Pins* grid. The pin type must be POWER, NC, or GROUND.
2. Click *Move – Logical Pins to Global*.

The pin appears in the *Global Pins* grid.

To move a pin from *Global Pins* to *Logical Pins*:

1. Select the global pin.
2. Choose *Move – Global Pins to Logical*.

The pin moves up to the *Logical Pins* grid. If the global pin was mapped to more than one physical pin, then the pin gets added to the logical pin list as a vector pin.

To alter existing pin mappings:

1. Unmap the pins whose mappings are to be modified.
2. Remap the pins as per the new requirements.

## Converting Scalar Pins to Vector

Two types of scalar pins can be converted to vector pins:

- Duplicate pins, such as multiple instances of GND when data is copied from PDF
- Scalar pins, such as A1, A2, A3 and so on

To convert:

1. Right-click on the scalar pin in the Package Editor and select the *Select To Make Bits* option.

This will select all scalar pins that can be converted to the bits of a vector pin. For example, if there are scalar pins A1, A2, and A3 and you selected the option on A1, then pins A2 and A3 will also get selected.

2. Right-click on the selection and choose *Convert To Bits*.

The scalar pins are converted to the bits of a vector pin.

**Note:** The scalar pin list is changed only for the selected package. Other packages and the master pin list will continue to have these pins as scalar pins.

## Modifying Package Pin Properties

You can modify package pin properties by deleting properties or by renaming properties.

### Deleting Package Pin Properties from All Pins of the Package

To delete package pin properties from all pins of the package, do the following:

1. Choose *Properties – Delete*.  
The Delete Package Pin Property dialog box appears.
2. Select the properties to be deleted and click *OK*.

### Deleting Package Pin Properties from Specific Pins

- Delete the value of the package pin property to remove the property from the pin.

## **Renaming Package Pin Properties**

To rename package pin properties, do the following:

1. Choose *Properties – Rename*.

The Rename Package Pin Property dialog box appears.

2. Specify the new names in the *New Name* column and click *OK*.

## **Deleting Symbol and Package Pins**

To modify a package, you may also want to delete its pins. You can delete a pin from a specific package or symbol, or from all packages and symbols.

To delete pins from a specific symbol or package, use the Symbol Editor or Package Editor. To delete a pin from all symbols and packages, use the *Pins* option. When you delete a pin from all symbols and packages, the pin is permanently deleted.

### **Deleting Symbol or Package Pins**

To delete a symbol or a package pin, do the following:

1. Select the symbol or the package in the Cell Editor.

The symbol information appears in the Symbol Editor.

To view the information on default Symbol Editor, [Symbol Editor User Guide](#).

The package information appears in the Package Editor.

2. Click *Symbol Pins* or *Package Pin* depending on what you want to delete.

3. Right-click on the pin to be deleted and select *Delete Selected Rows*.

**Note:** To select multiple pins for deletion, press Shift and click on the pins. Then, right-click and select *Delete Selected Rows*.

4. Choose *File – Save*.

The pin is deleted from the selected symbol or package.

If the pin is not part of any other symbol or package, Part Developer displays a warning which prompts you to delete the pin permanently by using the Add Pin dialog.

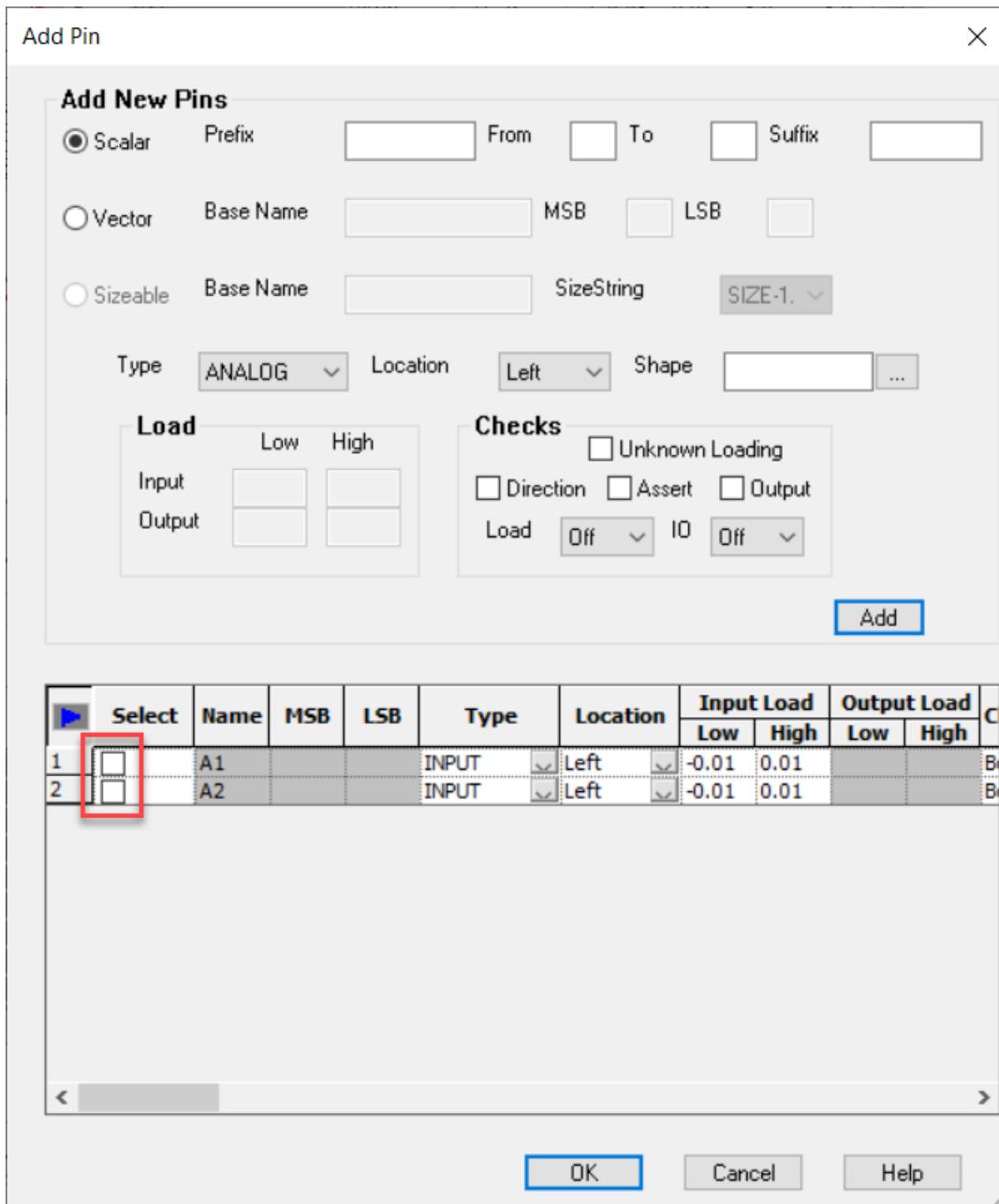
If you intend to use this pin in some other symbol or package or if you simply want to retain the pin information in the metadata, ignore the warning.

## Part Developer User Guide

### Modifying Parts

If you do not intend to use this pin elsewhere, use the Add Pin dialog to delete the pin. This deletes the pin permanently from all files of the cell.

To delete a pin from the Add Pin dialog, select the required check box in the Select column, and click *OK*.



## Pin Deletion across All Packages and Symbols

If you want to permanently delete a pin, you can delete it from all packages and symbols using the following procedure:

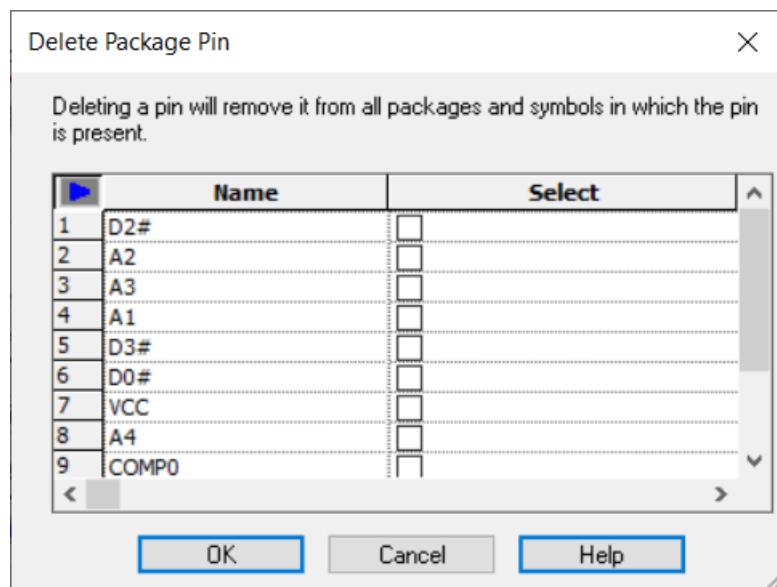
1. Select the package or symbol in which the pin is present.

The package or symbol information appears in the Package Editor or the Symbol Editor.

2. Click *Package Pin* or *Symbol Pins*.

3. Choose *Pins – Global Delete*.

The Delete Package Pin or Delete Symbol Pin dialog box appears. The dialog box lists all the pins in the package or on the symbol.



4. Select the pins to be deleted and click *OK*.

The pins will be deleted from all the symbols/packages.

## Modifying Symbols

Symbols are modified through the Symbol Editor. You can modify the following information:

- symbol properties and their attributes
- symbol texts and their attributes

- symbol pins. For details, see [Modifying Logical Pins](#) on page 155
- symbol outline
- symbol pins by moving them
- symbol pin properties

## Modifying Symbol Properties

Do the following to modify the symbol properties:

- To add symbol properties, press **Ctrl + I** in the *Properties* grid, add the symbol property and value, and determine its display attributes.
- To remove symbol properties, select the properties, right-click on the selection and select *Delete Selected Rows*.
- To modify an existing property and/or its display attributes, select the property and make the required changes.

## Modifying Symbol Text

Do the following to modify the symbol texts:

- To add symbol text, press **Ctrl + I** in the *Text* grid, and add the symbol text and determine its display attributes.
- To remove symbol texts, select the symbol texts, right-click on the selection and select *Delete Selected Rows*.
- To modify an existing text and/or its display attributes, select the text and make the required changes.

## Modifying Symbol Outline

- To modify the symbol outline, change the values of the left, right, top, and bottom fields.

## Move Pins

Do the following to move the symbol pins:

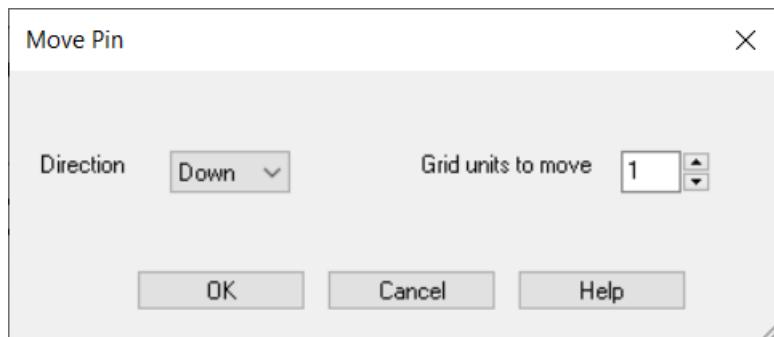
1. From the *Logical Pins* grid, select the pins.

2. Click the appropriate arrow button to move the selected symbol pins. The pins will move by one grid at a time.

Do the following to move the pins more than one grid at a time:

1. Click the *Move* button in the *Move Pins* group box.

The Move Pin dialog box appears.



2. Select the direction in which the pins are to be moved.
3. Specify the grid units by which to move and click *OK*.

## Modifying Symbol Pin Properties

You can modify symbol pin properties by deleting properties or by renaming properties.

### Deleting Symbol Pin Properties from all Pins of the Symbol

To delete symbol pin properties from all pins of the symbol, do the following:

1. Choose *Properties – Delete*.

The Delete Symbol Pin Property dialog box appears.

2. Select the properties to be deleted and click *OK*.

### Deleting Symbol Pin Properties from Specific Pins

- Delete the value of the symbol pin property to remove the property from the pin.

## **Renaming Symbol Pin Properties**

To rename symbol pin properties, do the following:

1. Choose *Properties – Rename*.

The Rename Symbol Pin Property dialog box appears.

2. Specify the new names in the *New Name* column and click *OK*.

## **Modifying the Attributes of a Symbol Pin Properties**

To modify the attributes of a symbol pin property, do the following:

1. Select the symbol pin property in the *Logical Pins* grid.

2. Select *Properties – Attributes*.

The Symbol Pin Properties dialog box appears.

3. Modify the attributes as required and click *Close*.

## **Modification Tips**

Part Developer provides several methods using which you can speed up part modification. These methods are described below.

### **Modifying a Value for Multiple Rows or Columns**

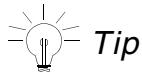
In Part Developer, you can modify the value of a field across multiple rows or columns in a single step by doing the following:

1. Select the rows for which data needs to be modified.

2. Right-click and select *Modify Values*.

The Modify Column Values dialog box appears.

3. Specify the new values and click *OK*.



You can also use the Modify Values feature directly when working with grid contents. For example, if you want to modify the pin type for a set of pins, you can select the pin type values for the pins in the grid, type the new pin type, and then press Enter. The new values appear for all the selected rows.

## Using Filters

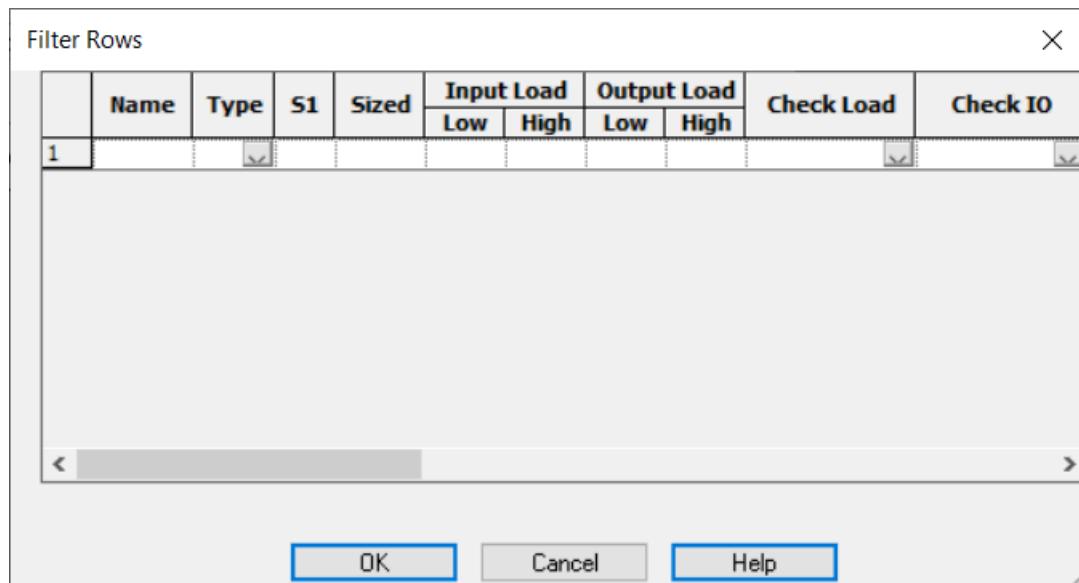
Often, a grid will contain many rows of data. For example, a large pin-count part may have thousands of pins appearing in the *Logical Pins* grid. Part Developer provides a mechanism by which you can set up a filter condition and view only those rows that match the filter criterion.

### Creating a Filter

To create a filter, do the following:

1. Right-click on the grid where you want to apply the filter and select *Filter Rows*. For example, suppose you want to view only the dip footprints in the Browse Jedec Type dialog box. So, right-click on any of the rows in the Browse Jedec Type dialog box.

The Filter Rows dialog box appears.



2. Specify the filter condition and click *OK*. For example, to see only the dip footprints, enter `dip*`.

Only those rows that match the filter criterion appear in the grid.

**Note:** When a filter is applied, the filter viewer icon appears in green. This provides a visual indication that some rows or columns are hidden in the grid.

### **Removing a Filter**

- To remove a filter, right-click on the grid and select *Unhide All Rows*.

### **Using Find Filter to Locate Symbol Objects**

The *Find* tab in the Symbol Editor enables you to quickly search symbol objects for modifying them using the Symbol Editor.

For more information about the options on the *Find* page, see [Find](#) on page 64.

## **Part Developer User Guide**

### Modifying Parts

---

---

## Creating Shapes

---

While building parts, a significant amount of time is typically spent on symbol graphic creation. Creating commonly used graphics, such as IEC and IEEE shapes to represent pins and functions such as ALU, is time-consuming. Although you can copy graphics from one symbol and paste them in another symbol, this method of reusing graphics is not quite intuitive and efficient.

A better way is to create reusable library elements of pin and custom shapes. These shapes enable you to build symbols with graphics that illustrate the functionality of the symbols according to the standards and requirements of your company. By attaching shapes to symbol pins, you can create symbols that help users understand at a glance the functions of various pins.

You can build your own shape library over a period of time and optimize the symbol creation process by enabling the reuse of shapes. You can even extract shapes from legacy symbols and reduce the shape library development time. Conversely, you can update the legacy symbols with new and reusable shapes.

Options to add or modify shape information are now available from Add Pin, Modify Pin, and Symbol Pin Attributes dialog boxes. CSV import is also enhanced to support the import of pin shapes from .csv files.



After a shape is attached to a symbol, any graphical changes made to the shape by using the Symbol Editor do not constitute an integral part of the shape. To edit a shape, you should use the Shape Editor and then re-attach the shape to the symbol.

To view the information on default Symbol Editor, [Symbol Editor User Guide](#).

## Types of Shapes

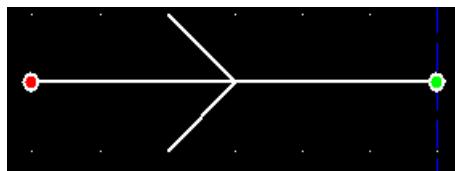
In Part Developer, shapes are classified into the following types:

- Pin shapes

■ Custom shapes

## Pin Shapes

Pin shapes are shapes that you associate with symbol pins. A pin shape must have a connection point, which is the point where a connection can be made with the pin, and an anchor point, which is the point where the shape is connected to the symbol.



In the above graphic, the green spot indicates the anchor point whereas the red point indicates the connection point. The distance between the anchor point and the connection point is called the stub length.

**Note:** It is recommended that the stub length you specify when creating a pin shape is same as the stub length specified in Setup.

## Custom Shapes

Custom shapes are any shapes other than pin shapes and can contain special characters and text. They are primarily used for documentation purposes and can be associated with pins or the complete symbol. Any custom shape that is placed directly on a symbol is treated as a custom shape associated with that symbol. You can use a custom shape either independently or as part of other shapes in a symbol.

Unlike a pin shape, a custom shape does not have a connection point. Its anchor point determines its placement with respect to the origin of the symbol.



## Shape Editor

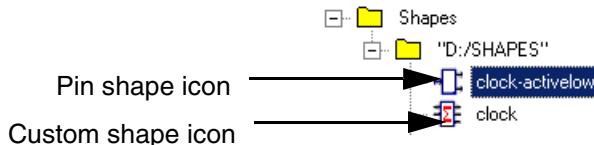
In the PCB Librarian XL suite, Part Developer provides the Shape Editor to help you create and modify shapes, which you can add to symbols to illustrate the functionality of the symbols.

The Shape Editor is divided into the following three panes:

- The Shape Editor tree pane
- The Shape Attributes pane
- The Shape Editor canvas

### Shape Editor Tree

The Shape Editor tree pane displays the names of the various shapes in your shape library in a tree structure. The paths from which shapes are loaded in the current project are displayed under *Shapes*, which is the top node of the tree. The icons displayed to the left of shape names indicate if a shape is a custom shape or a pin shape.



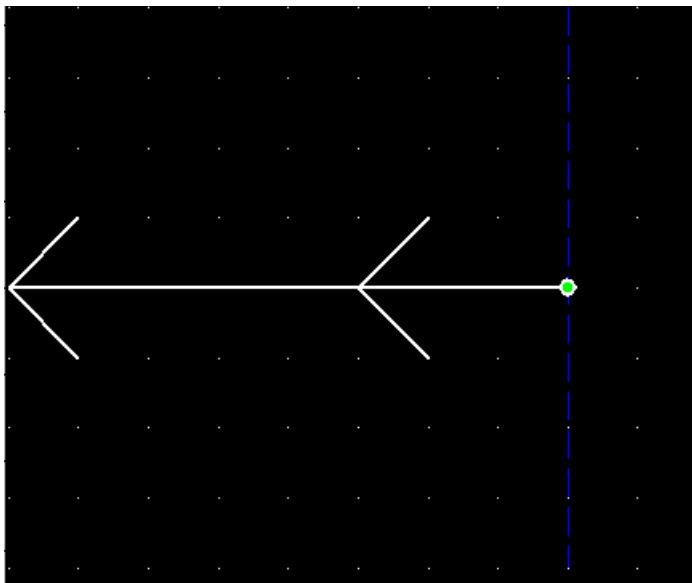
### Shape Attributes Pane

The Shape Attributes pane displays the attributes of the shape selected in the Shape Editor tree pane. You can modify the shape type and reset the anchor point. For a pin shape, you can also set the connection point. If you want that the shape should be attached to only certain sides of a symbol, you can use the *Left*, *Right*, *Top*, and *Bottom* check boxes to specify your requirement. By default, all shapes can be placed on any of the four sides.

The *Set \$PN Location* button enables you to specify a location on a pin shape for displaying the value of the *\$PN* property.

## Shape Editor Canvas

The Shape Editor canvas, which is similar to the Symbol Editor canvas, displays the shape selected in the Shape Editor tree pane.



The Shape Editor canvas has a blue line on which a bright green spot appears. This line simulates the symbol-bounding line for pins that are attached to the left of a symbol. The bright green spot indicates the anchor point for the pin, which is used as the origin for various operations.

## Shape Editor Grid Settings and Behavior

Like the Symbol Editor, the Shape Editor supports two types of grids. By default, a coarse grid is displayed. You can use the following tool buttons to hide grid lines, display the coarse grid, or display the finer grid:



Hides grid lines or  
displays the coarse  
grid



Displays the finer  
grid

The connection point of a pin shape is always placed on the pin grid.

For information on changing the density of the finer grid, see [Non-pin grid factor](#) on page 93.

### Moving Objects on the Shape Editor Canvas Using Arrow Keys

You can use the arrow keys on your keyboard to move objects on the editing canvas. Moving of all objects on the Shape Editor Canvas by using arrow keys depends on the visible grid and the selection of *Snap to Grid* as tabulated below:

Grid and Snap to Grid Selection	Shape Objects
Default pin grid visible and <i>Snap to Grid</i> selected	Move on the non-pin grid
Non-pin grid visible and <i>Snap to Grid</i> selected	Move on the non-pin grid
Default pin grid visible and <i>Snap to Grid</i> deselected	Nudge
Non-pin grid visible and <i>Snap to Grid</i> deselected	Nudge

## Setting Up a Project for Shapes

Before you create and modify shapes, you need to set up a project in the following way:

1. Create a new library project or open an existing library project.
2. In *Tools – Setup*, click the *Shape* node.
3. In the *Shape Path* field, specify the path from which Part Developer needs to load shapes.
4. In the *Default Shape Save Path* field, specify the path in which newly created pin shapes and custom shapes are to be saved.

**Note:** The *Shape Path* field is always in append mode. The new path you specify gets appended to the existing path with a semicolon used to separate the new path and the existing path.

5. Use the *Grid Size* option to define the Shape Editor grid settings. By default, the grid size is 0.05 inches, which is the same as the default size of the Symbol Editor grid. You can change the value as well as the unit according to your requirements.
6. The *Maximum Size (In Grid Units)* option defines the pin shape height and width when the shape is attached to a pin. The default values for both width and height are 0 symbol-grid units. The default values retain the size of the actual pin shape when the shape is added to a pin in a symbol. Specifying any other value stretches or shrinks the pin shape according to the specified values for width and height.
7. Close the project and reload it to reflect the changed settings.

## Creating a Shape

To create a custom shape or a pin shape:

1. Choose *File – New – Shape*.
2. Depending on whether you want to create a custom shape or a pin shape, choose *CustomShape* or *PinShape*.
3. In the New Custom Shape or New Pin Shape dialog box, specify the required shape name and click *OK*.

The Shape Editor opens with the new shape as the active shape. All of the available custom shapes and pin shapes are displayed in a tree view in the left pane.

4. Create the shape by using the standard graphical toolbars or *Graphic Editor* menu options.  
For information on various *Graphic Editor* menu options, see [Graphic Editor](#) on page 31.
5. To set the connection point for a pin shape, click the *Set ConnectionPoint* button in the Shape Attributes pane, take the mouse pointer to the Shape Editor canvas, and click at the desired location.
6. In a similar way, set the location of \$PN for a pin shape by using the *Set \$PN Location* button in the Shape Attributes pane.
7. For both pin shapes and custom shapes, if required, reset the anchor point by using the *Set AnchorPoint* button in the Shape Attributes pane.
8. Save the shape.

Part Developer validates the shape and report errors if any.

## Extracting Shapes from Existing Symbols

To extract a custom shape or a pin shape from an existing symbol:

1. Open the cell and display the symbol on the Symbol Editor canvas.
2. Select the graphical object that you want to extract as a shape by drawing a rectangle around the object.
3. Right-click and choose *Extract Shape*.

Part Developer prompts for the name of the shape.

4. Specify the name and click *OK*.

The selected shape is extracted in the Shape Editor.

You can edit the shape and save it in the same way as you save any shape.

**Note:** By default, the type of the extracted shape is *CustomShape*. If you want to save the extracted shape as a pin shape, you need to change the type to *PinShape* and specify the connection point by using the *Set ConnectionPoint* button.

## Replacing Symbol Pin Shapes

You can replace symbol pin shapes by using either of the following two methods. However, the advantage of the first method is that it lets you to select not only the pin shape but also the objects around the pin and replace all of the selected objects in a single operation.

### Method 1

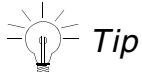
To replace the shape of an existing symbol pin with another shape:

1. Open the cell and display the symbol in the Symbol Editor.
2. Select the pin for which you want to change the pin shape.
3. Right-click and choose the *Replace PinShape* option from the pop-up menu.

The Shape Viewer appears.

4. Choose the required shape and click *OK*.

The new shape is attached.



**Tip**

To replace the shapes of multiple pins, first select the required pins and then choose the *Replace PinShape* option.

## Method 2

To attach a pin shape to an existing symbol pin:

1. Open the cell and display the symbol on the Symbol Editor canvas.

2. Choose *Pins – Attributes*.

The Symbol Pin Attributes dialog box opens.

3. Choose the *Select...* option from the *Shape* drop-down list and select the required pin shape from the Shape Viewer.

4. Click *OK*.

You will notice that the new pin shape is attached.

**Note:** All grid features, such as multi-select and Modify Values, work as usual.

## Attaching Custom Shapes to Existing Symbol Pins

To attach custom shapes to existing symbol pins:

1. Open the cell and display the symbol in the Symbol Editor.

2. Choose *Pins – Attributes*.

The Symbol Pin Attributes dialog box opens.

3. Click in the *Shape 1* cell—under the Custom Shape column heading—for the symbol pin you want to modify.

4. To display the Shape Preview dialog box, click the down arrow button in the *Shape 1* cell.

5. Select the custom shape name from the list of shape names displayed in the left pane.

The selected custom shape is displayed on the Shape Preview canvas.

6. Click *OK*.

7. If required, attach another custom shape to the same pin by using the *Shape2* cell under the Custom Shape column heading.

**8. Click Close.**

**Note:** You can attach a maximum of two custom shapes to a symbol pin.

## Inserting Custom Shapes

To insert a custom shape into a symbol:

1. Open the cell and display the symbol in the Symbol Editor.
2. Right-click on the Symbol Editor canvas.
3. Choose the *CustomShape* option.

The Shape Preview dialog box displays.

4. Click the shape name and click *OK*.

Notice that the mouse pointer changes to indicate that a shape is being inserted as in the case of adding an image in the Symbol Editor.

5. Click at the location on the Symbol Editor canvas where the shape is to be inserted.

## Aligning Custom Shapes

When you move a custom shape from one location on the Symbol Editor canvas to another with the *Snap to Grid* option selected, by default, the lower left point of the rectangular boundary of the custom shape is placed on the grid.

Part Developer enables you to change the default setting while editing a symbol and use one of nine points specified with respect to the rectangular boundary of a custom shape as the align point. The following table lists the available align points and the key combinations you can use to select them:

---

Align Point	Key Combination
Upper left	Ctrl + 1
Upper center	Ctrl + 2
Upper right	Ctrl + 3
Center left	Ctrl + 4
Center	Ctrl + 5

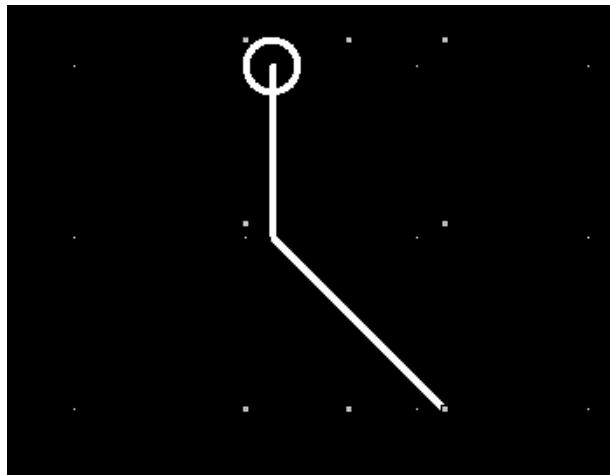
## Part Developer User Guide

### Creating Shapes

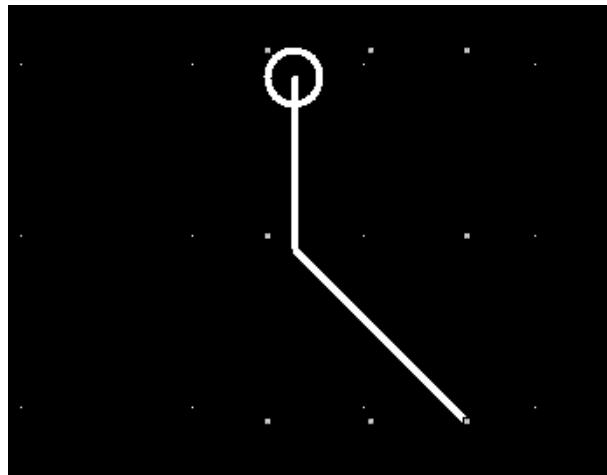
---

Align Point	Key Combination
Center right	Ctrl + 6
Lower left	Ctrl + 7
Lower center	Ctrl + 8
Lower right	Ctrl + 9

In the following graphic, a custom shape is aligned in two ways, first in the default way and then with respect to the center left point on its rectangular boundary. Notice that only the lower left point of the rectangular boundary is on grid in the default way of alignment whereas only the center left point is on grid in the other type of alignment.



Aligned with respect to the lower left point



Aligned with respect to the center left point

---

# Working with VHDL Wrappers and Map Files

---

Design Entry HDL generates a netlist that you can use to simulate your logical design using a Verilog- or VHDL-based simulator. To generate the netlist, it uses the symbol pin names as the interface ports for each component used in the design. For each component, the symbol pin names are saved as ports in a VHDL entity declaration file in the entity view of the part. The entity declaration from the symbols may be different from the actual VHDL model's entity declaration. Therefore, to successfully simulate a design, it is required that the port names generated for a component in the HDL netlist match the port names in the actual VHDL model. For example, for the part 74ac74, the port names in the entity view are as follows:

```
-- generated by newgenasym Wed Oct 30 15:34:37 2002
```

```
library ieee;
use      ieee.std_logic_1164.all;
use      work.all;
entity \74ac74\ is
  port (
    CLK:      IN      STD_LOGIC;
    D:        IN      STD_LOGIC;
    Q:        OUT     STD_LOGIC;
    \|clr|\*: IN      STD_LOGIC;
    \|pre|\*: IN      STD_LOGIC;
    \|q|\*:   OUT     STD_LOGIC);
end \74ac74\;
```

For the same part, the VHDL model's port names or entity are:

```
library ieee;
```

## Part Developer User Guide

### Working with VHDL Wrappers and Map Files

---

```
use ieee.std_logic_1164.all;

entity sn74ac74 is
    generic (
        tw_cntl      : time := 5.002 ns; -- min pulse duration for pre* and clr*
        tw_clk       : time := 50 ns; -- min pulse duration for clk
        tsu_cntl     : time := 0 ns; -- max setup time pre* and clr*
        ts_data      : time := 3 ns; -- max setup time for data
        th_data      : time := 0.5 ns; -- max hold time for data
        tp_cntl_max : time := 10.5 ns; -- max prop delay for cntl to Q
        tp_clk_max  : time := 10.5 ns -- max prop delay for data to Q
    );
    port (
        pre1_n : in std_logic := 'U';
        clr1_n : in std_logic := 'U';
        d1      : in std_logic := 'U';
        clk1   : in std_logic := 'U';
        q1      : out std_logic;
        qb1    : out xyz;
        pre2_n : in std_logic := 'U';
        clr2_n : in std_logic := 'U';
        d2      : in std_logic := 'U';
        clk2   : in std_logic := 'U';
        q2      : out std_logic;
        qb2    : out std_logic
    );
end sn74ac74;
```

So, the mechanisms that are employed to map the symbol pin names (or port names in the entity view) with the port names in the VHDL model are files called wrappers and map files. The wrappers or map files can be created using Part Developer.

## Creating a VHDL Map File

VHDL map files can be created using Part Developer. When creating a VHDL wrapper or map file, specify the following information:

- For VHDL map files, specify the package. The package information is required to determine the logical pin list, the number of slots, slots in which a logical pin list is present, the pin mode, and the pin type.
- The path to the VHDL model. This can be done by specifying either the actual physical path to the VHDL model or the Lib:Cell:View structure.

For example, to access the VHDL model stored in x:~~•~~74ac:sn74ac74, you may provide either the entire physical path or use the lib:cell:view method, i.e., specify 74ac:sn74ac74::vhdl\_lib.

**Note:** To use the lib:cell:view method, there should be a library entry in the `cds.lib` file of the project. For example, to use the sn74ac74 with the lib:cell:view method, the following entry should be present in the `cds.lib` file:

```
DEFINE 74ac x:/74ac
```

- Determine whether to bind the VHDL model and the symbol. That is, determine whether to bind the symbol with one specific architecture (behavior). If you decide to bind the VHDL model and the symbol, then the binding statement goes into the wrapper. This is an optional step.

**Note:** Since each view of the model is essentially a specific architecture, selecting the VHDL model by using the lib:cell:view method automatically fills in the binding statement. If you specify the actual physical path to the VHDL model, then you have to explicitly enter the binding information. If you decide not to enter the binding information during wrapper creation, then the binding information has to be specified later in the simulation flow. However, not providing the binding information in the wrapper provides the freedom to use the same wrapper/map file for different architectures.

- By default, all the generics present in the VHDL model gets written into the map/wrapper file. You need to determine the generics and their values to annotate to the symbol.
- Enter the mappings between the logical pins and the model ports.

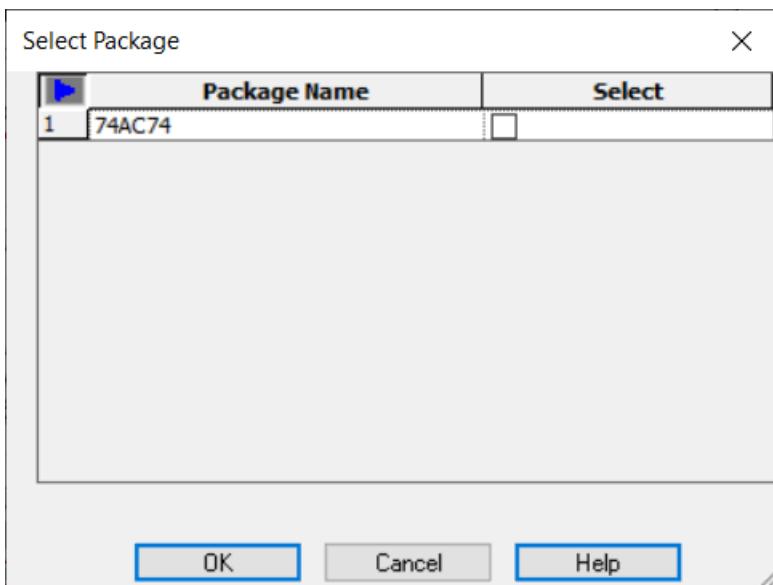
The 74ac74 part is used to demonstrate the steps in creating a VHDL map file.

1. Choose *File – New – VHDL Map File*.

## Part Developer User Guide

### Working with VHDL Wrappers and Map Files

The *Select Package* dialog box appears. All packages and aliases are listed separately.

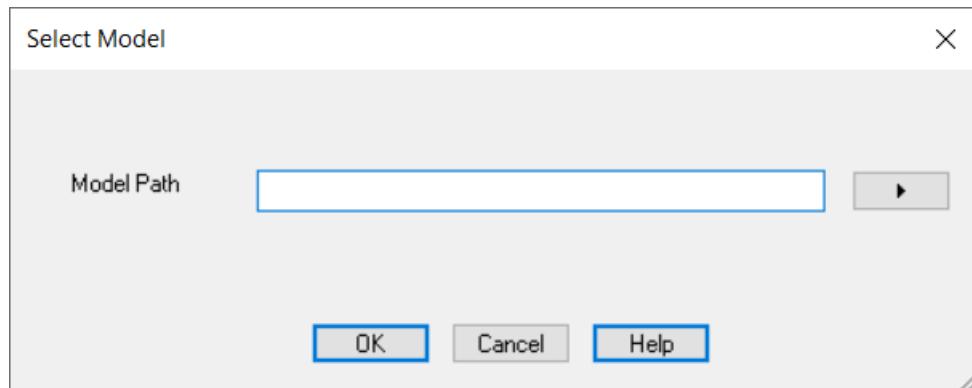


2. Select one or more packages and click *OK*. The pin list of the selected packages is used for the mapping.

 **Important**

Multiple packages can be selected only if the packages meet certain criteria. See [Select Package](#) on page 423 for more details.

The Select Model dialog box appears.



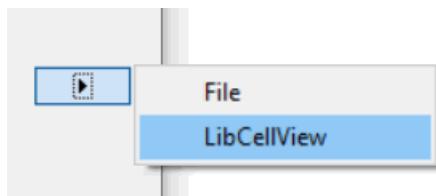
3. Specify the path to the VHDL model in the *Model Path* field. To browse to the VHDL model location, use either the File browser or the Lib:Cell:View browser. The Lib:Cell:View method is explained here.

## Part Developer User Guide

### Working with VHDL Wrappers and Map Files

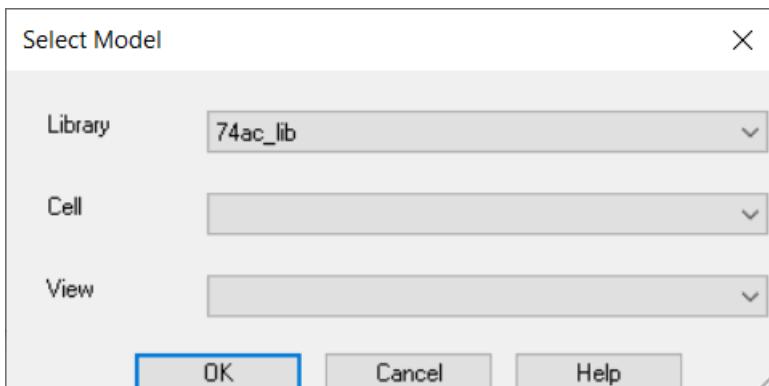
4. To launch the file browser or the lib:cell:view option, click on the button with the arrow.

A pop-up menu with *File* and *LibCellView* options appears.



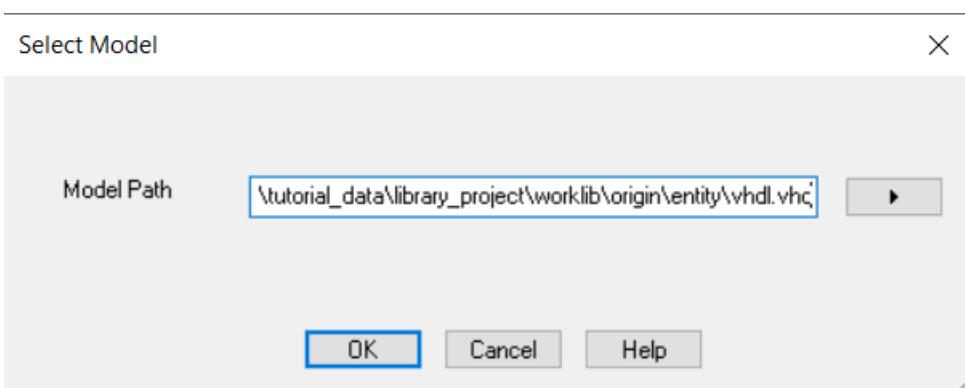
5. Select *LibCellView*.

The Select Model dialog box appears.



6. Select the *Lib:Cell:View* location where the VHDL model is stored and click Ok.

The Select Model dialog box appears with the path to the VHDL model seeded in the *Model Path* field.

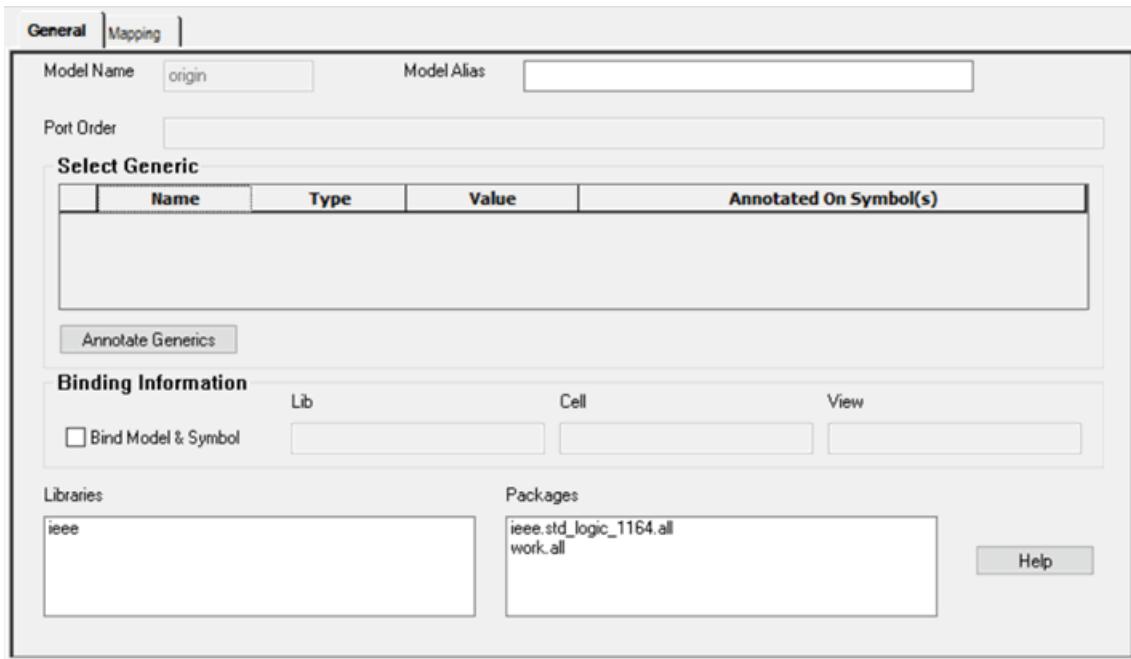


7. Click *Ok*.

## Part Developer User Guide

### Working with VHDL Wrappers and Map Files

The *VHDL Map File* editor appears in the Cell Editor.



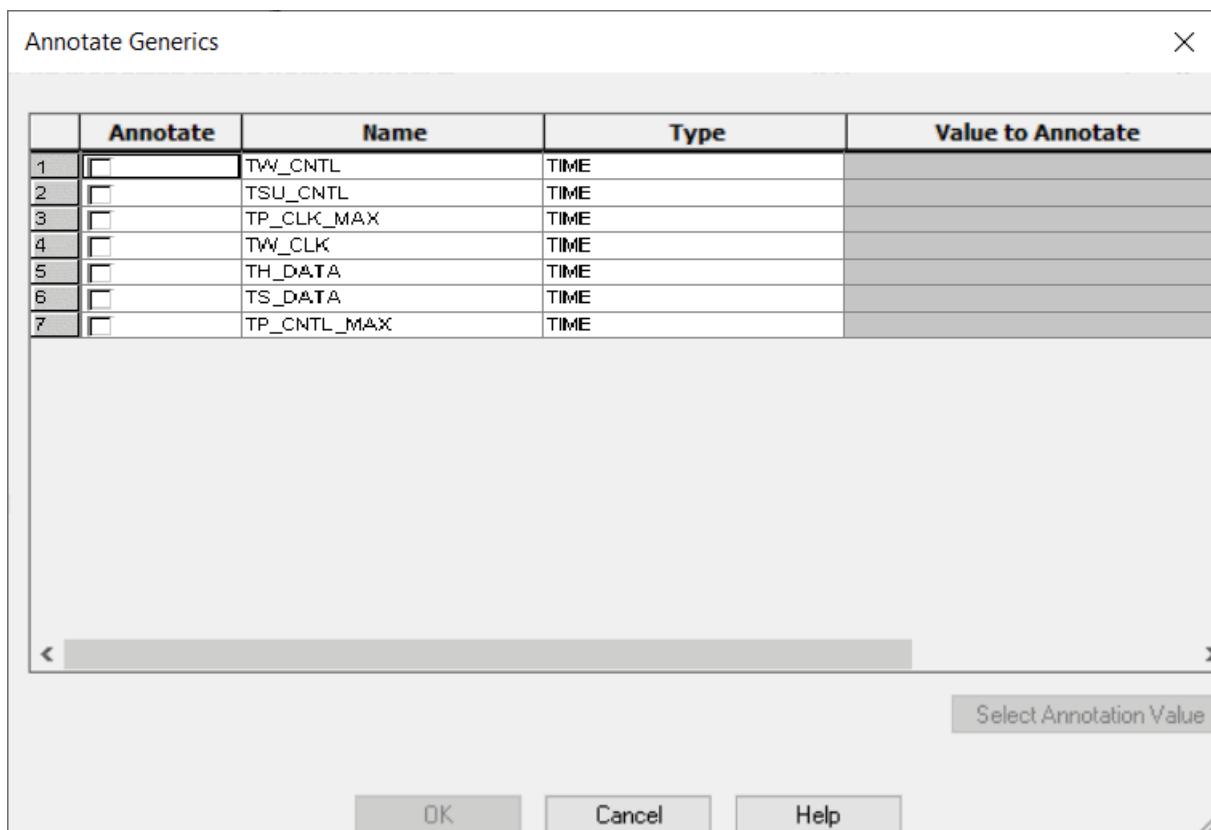
The *General* page appears in the right pane. There are four non-editable fields in this page: *Model Name*, *Port Order*, *Libraries*, and *Packages*. The *Model Name* field displays the name of the VHDL model and the *Port Order* field displays the exact order of ports as they appear in the entity. The *Libraries* and the *Packages* field shows the libraries that needs to be included for the VHDL model file to compile and simulate. The library and the package list are read from the VHDL model file.

8. Specify the aliases for the model in the *Model Alias* field.
9. The *Select Generic* group box displays the generics that exist in the VHDL model. By default, all the generics present in the VHDL model are added to the map file, but none of these generics are annotated on the symbols. To annotate a generic, click *Annotate Generics*.

The *Annotate Generics* dialog box appears. In addition to the generics present in the selected VHDL model, this dialog box will also show all the generics that are present in all the VHDL map/wrapper files for the part.

## Part Developer User Guide

### Working with VHDL Wrappers and Map Files



10. Select the generics that you want to add to the symbols and click *Ok*.

**Note:** When you select a generic, the value appears in the *Value to Annotate* field. By default, the value is the one that exists for the generic in the VHDL model. In case you want to change the value, click *Select Annotation Value*. This will display the Select Value to be Annotated dialog box through which you can specify the value for the selected generic. If a generic is present in multiple wrappers/map files with different values, then all the values are displayed for the generic through a drop-down list. You can either select one of the existing values from the list or specify a new value.

11. Determine whether to bind the VHDL model and the symbol. That is, determine whether to bind the symbol with one specific architecture (behavior). If you decide to bind the VHDL model and the symbol, then the binding statement goes into the wrapper.

Next, you need to do the mapping of the VHDL model port list with the logical pin list. The mapping is done through the *Mapping* page.

12. Click *Mapping*.

## Part Developer User Guide

### Working with VHDL Wrappers and Map Files

The *Mapping* page appears.

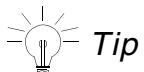
The screenshot shows the 'Mapping' tab selected in the top navigation bar. Below it are two tables: 'Logical Pin List' and 'Model Port List'. The 'Logical Pin List' table has columns for Name, Section, Mode, and Type. The 'Model Port List' table has columns for Name, Mode, Type, Logical Pin, and Function. Between the two tables are several buttons: 'Map', 'Auto Map', 'Unmap', 'Unmap All', and 'Help'. At the top of each table is a checkbox for 'Automatically Update Pin Mode' and another for 'Automatically Update Pin Type'.

Name	Section	Mode	Type
2 CLK	S1	IN	STD_LOGIC
3 CLR1*		IN	STD_LOGIC
4 D		IN	STD_LOGIC
5 PRE1*		IN	STD_LOGIC
6 Q		OUT	STD_LOGIC
7 Q1*		OUT	STD_LOGIC

Name	Mode	Type	Logical Pin	Function
1 PRE1_N	IN	STD_LOGIC		
2 CLR1_N	IN	STD_LOGIC		
3 D1	IN	STD_LOGIC		
4 CLK1	IN	STD_LOGIC		
5 PRE2_N	IN	STD_LOGIC		
6 CLR2_N	IN	STD_LOGIC		
7 D2	IN	STD_LOGIC		
8 CLK2	IN	STD_LOGIC		
9 Q1	OUT	STD_LOGIC		
10 GB1	OUT	XYZ		
11 Q2	OUT	STD_LOGIC		
12 QB2	OUT	STD_LOGIC		

The *Mapping* page displays the logical pin and the model port list. The *Logical Pins* grid displays the logical pins present in the selected packages, their modes, and the slots in which they are present. The slots for which a pin is absent is disabled. The pin mode is determined by the value of the `VHDL_MODE` property on the symbol pins. If the property is not found, then the `chips.prt` is read to determine the pin mode.

#### 13. Do the mappings as required.



The *Automap* button provides an automatic way of mapping the model ports to logical pins. See [Auto Map](#) on page 70 for details.

**Note:** The order in which you select the cells or rows will be the order in which the mapping is done. For example, for the logical pin CLK1, if you select the slot 2 first and S1 next, and then from the VHDL model ports, select CLK1 and CLK2 in that order, then on mapping, CLK1 will get mapped to slot S2 and CLK2 will get mapped to slot S1.

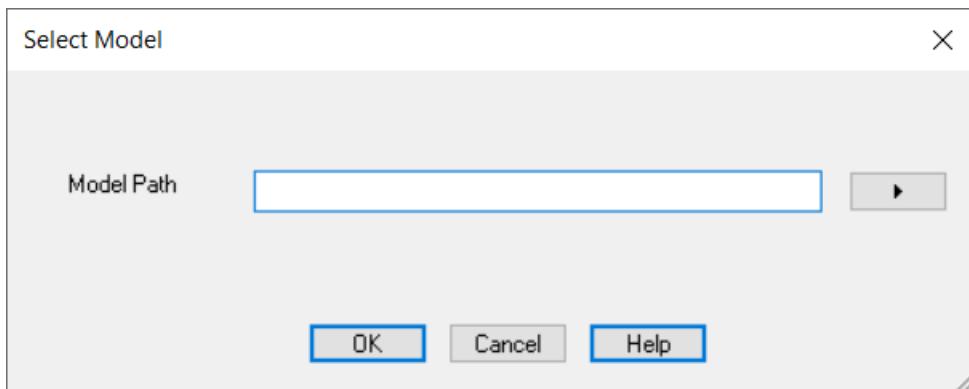
**Note:** In case you are mapping a symbol pin to a model port of different modes or pin types, then you need to check the Automatically Update Pin Mode and/or Automatically Update Pin Type options.

This completes the creation of a VHDL map file.

## Creating a VHDL Wrapper File

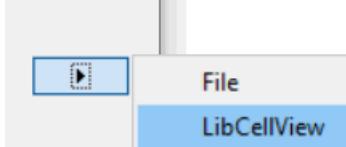
1. Choose *File – New – VHDL Wrapper*.

The Select Model dialog box appears.



2. Specify the path to the Verilog model in the *Model Path* field. To browse to the VHDL model location, use either the File browser or the Lib:Cell:View browser. The Lib:Cell:View method has been explained here.
3. To launch the file browser or the lib:cell:view option, click on the button with the arrow.

The *File/LibCellView* shortcut menu appears.



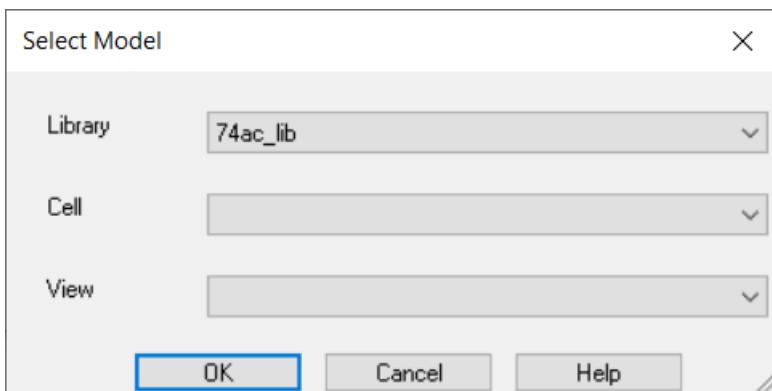
4. Select *LibCellView*.

## Part Developer User Guide

### Working with VHDL Wrappers and Map Files

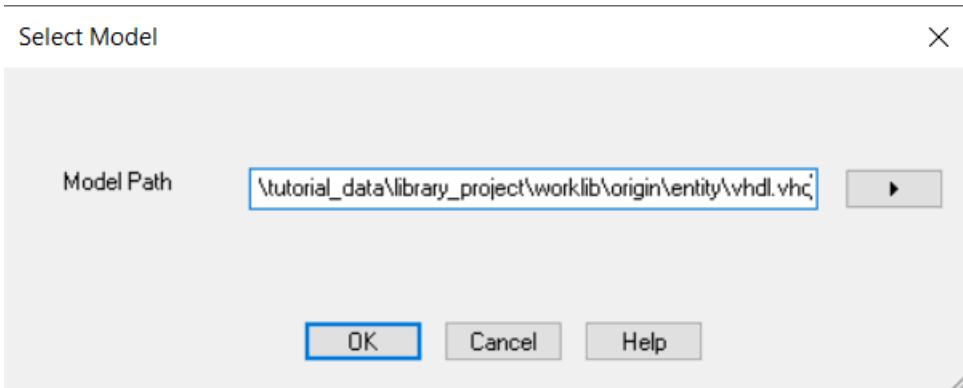
---

The *Select Model* dialog box appears.



5. Select the *Lib:Cell:View* location where the VHDL model is stored and click *Ok*.

The Select Model dialog box appears with the path to the VHDL model seeded in the *Model Path* field.

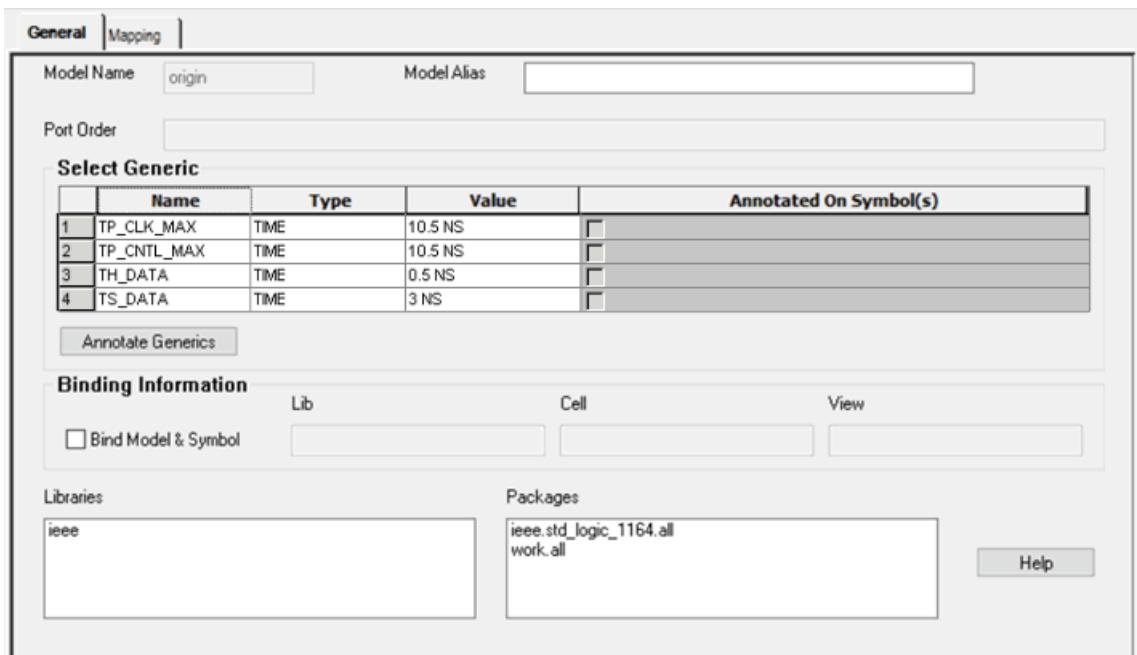


6. Click *Ok*.

The *VHDL Wrapper File* editor appears in the Cell Editor.

## Part Developer User Guide

### Working with VHDL Wrappers and Map Files



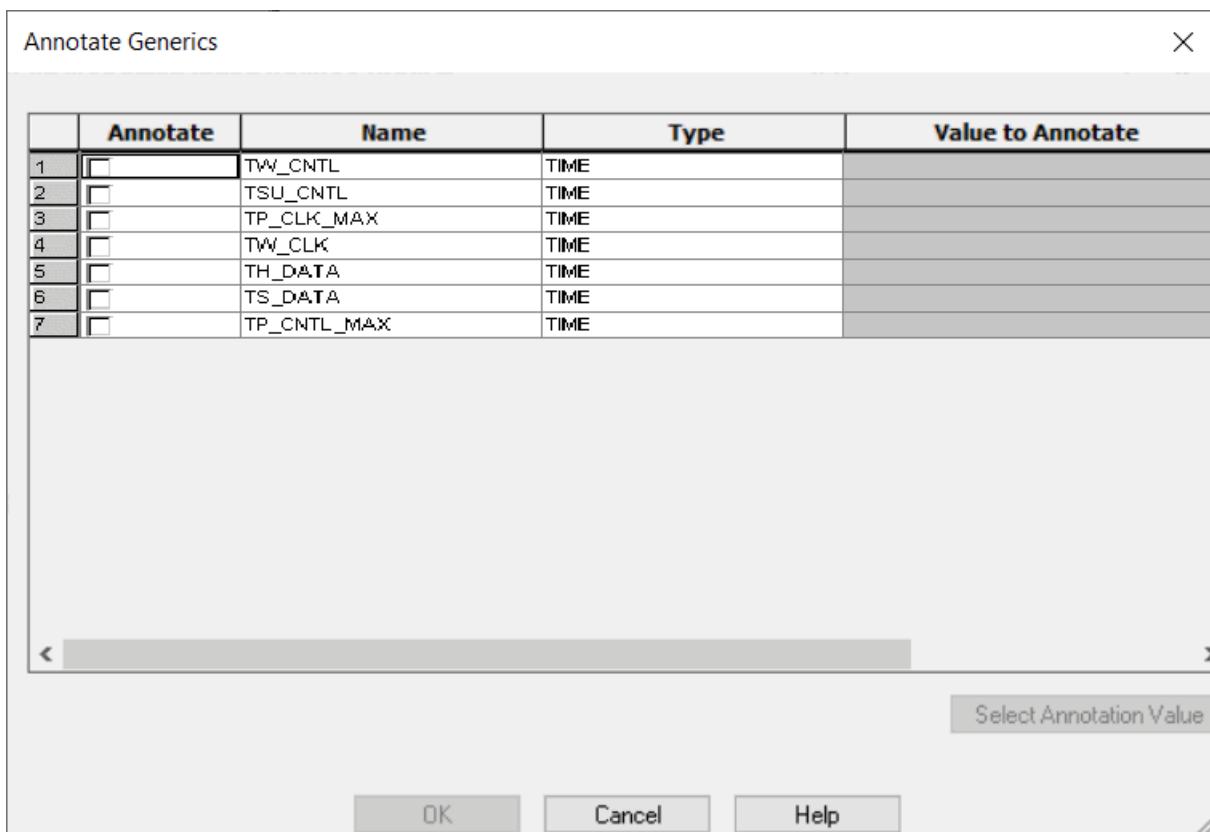
The *General* page appears in the right pane. There are three non-editable fields on this page: *Model Name*, *Libraries*, and *Packages*. The *Model Name* field displays the name of the VHDL model. The *Libraries* and the *Packages* list box shows the libraries that needs to be included for the VHDL model file to compile and simulate. The library and the package list are read from the VHDL model file.

7. The *Select Generic* group box displays the Generics that exist in the VHDL model. By default, none of these generics is annotated on the symbols. To annotate a generic, click *Annotate Generics*.

The Annotate Generics dialog box appears.

## Part Developer User Guide

### Working with VHDL Wrappers and Map Files



8. Select the generics that you want to add to the symbols and click *Ok*.

**Note:** When you select a generic, the value appears in the *Value to Annotate* field. By default, the value is the one that exists for the generic in the VHDL model. In case you want to change the value, click *Select Annotation Value*. This will display the Select Value to be Annotated dialog box through which you can specify the value for the selected generic. If a generic is present in multiple wrappers/map files with different values, then all the values are displayed through a drop-down list. You can either select an existing value or specify a new value.

9. Determine whether to bind the VHDL model and the symbol. That is, determine whether to bind the symbol with one specific architecture (behavior). If you decide to bind the VHDL model and the symbol, then the binding statement goes into the wrapper.

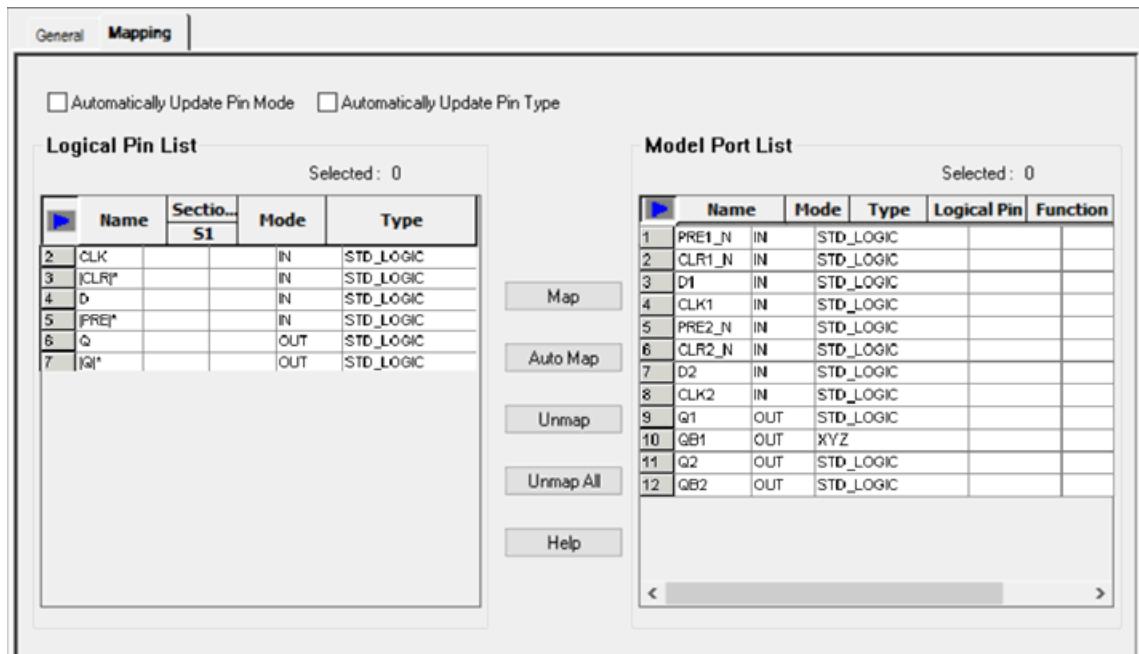
Next, you need to do the mapping of the VHDL model port list with the logical pin list. The mapping is done through the *Mapping* page.

10. Click *Mapping*.

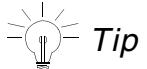
## Part Developer User Guide

### Working with VHDL Wrappers and Map Files

The *Mapping* page appears.



11. To map the logical pin names to VHDL model ports, select the logical pins in the *Logical Pin List* and VHDL model port in the *Model Port List* and click *Map*.



The *Automap* button provides an automatic way of mapping the model ports to logical pins. See [Auto Map](#) on page 70 for details.

**Note:** The order in which you select the rows will be the order in which the mapping is done. For example, if you select logical pin *CLK* and */CLR/\** from the *Logical Pins List*, and then select *CLK1* and *CLR1\_N* in that order from the VHDL *Model Ports List*, then on mapping, *CLK* will get mapped to *CLK1* and */CLR/\** will get mapped to slot *CLR1\_N*.

**Note:** In case you are mapping a symbol pin to a model port of different modes or pin types, then you need to check the *Automatically Update Pin Mode* and/or *Automatically Update Pin Type* options.

This completes the creation of a VHDL wrapper file.

## Modifying a VHDL Wrapper/Map File

Due to certain reasons, such as to change the binding information or generic informations, you may need to modify the existing VHDL wrappers. Using Part Developer, you can modify existing VHDL map files. Right-click on the VHDL wrapper that is to be modified.

1. Select the VHDL map file.

The VHDL Map File editor appears in the right pane of the Cell Editor window.

2. Change the values as required on the General and Mapping pages.
3. Select *File – Save* to save the changes.

## Deleting a VHDL Wrapper/Map File

To delete a VHDL wrapper:

1. Select the VHDL wrapper to be deleted.
2. Press **Delete**.
3. Click **Yes** to confirm the deletion.

## Renaming a VHDL Wrapper/ Map File

To rename a VHDL wrapper:

1. Right-click on the VHDL wrapper to be renamed.
2. Select *Rename*.
3. Enter the name for the wrapper.
4. Click **OK**.

---

# Creating Verilog Wrappers and Map Files

---

To ensure that Verilog XL can simulate the parts created using Part Developer, a mapping between the pin names of the part with the corresponding ports in the Verilog model is required. This mapping information is stored in a Verilog file termed Verilog wrapper or map file. The pin names in the wrappers are read from the entity view and the port names from the Verilog model.

**Note:** The entity view gets created automatically when the symbol view is created.

You can create the wrappers using Part Developer.

The LS241 part will be used to detail the steps to work with Verilog wrappers.

## Creating a Verilog Map File

Verilog wrappers or map files can be created using Part Developer. When creating a Verilog wrapper or map file, specify the following information:

- For Verilog map files, specify the package. The package information is required to determine the logical pin list, the number of slots, slots in which a logical pin list is present, and the pin mode.
- The path to the Verilog model. This can be done by specifying either the actual physical path to the Verilog model or the Lib:Cell:View structure.

For example, to access the Verilog model stored in the x:`74ac:sn74ac74/entity` location, you may provide either the entire physical path or use the lib:cell:view method, i.e. specify `74ac:sn74ac74:entity`.

**Note:** To use the lib:cell:view method, there should be a library entry in the `cds.lib` file of the project. For example, to use the `sn74ac74` with the lib:cell:view method, the following entry should be present in the `cds.lib` file:

```
DEFINE 74ac x:<74ac
```

## Part Developer User Guide

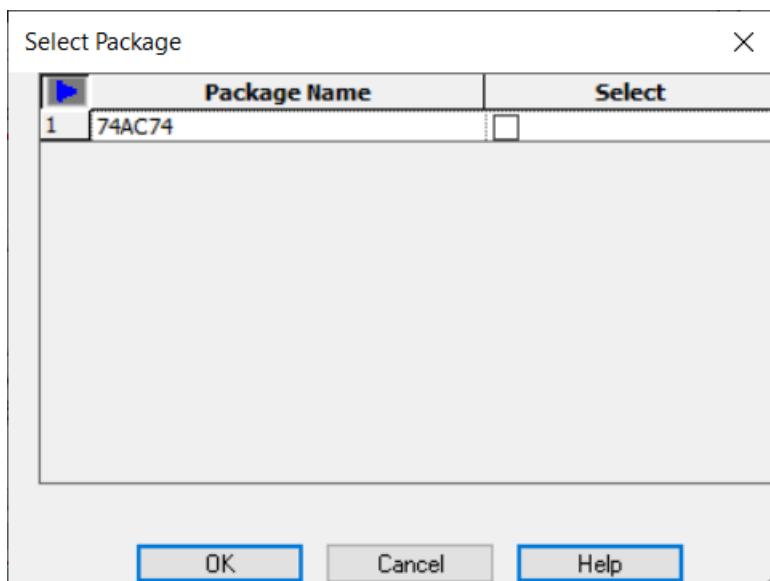
### Creating Verilog Wrappers and Map Files

- By default, all the parameters present in the Verilog model gets written into the map/wrapper file. You need to determine the parameters and their values to annotate to the symbol.
- Enter the mappings between the logical pins and the model ports.

The Dip package of the LS241 part is used to demonstrate the steps in creating a Verilog map file.

1. Choose *File – New – Verilog MapFile*.

The *Select Package* dialog box appears. All packages and aliases are listed as separately.



2. Select one or more packages. The pin list of the selected packages is used for the mapping and click OK.

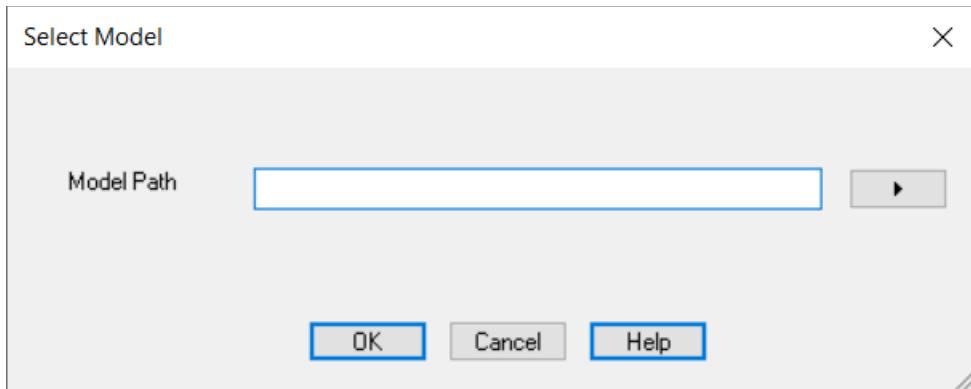


Multiple packages can be selected only if the packages meet certain criteria. See [Select Package](#) on page 423 for more details.

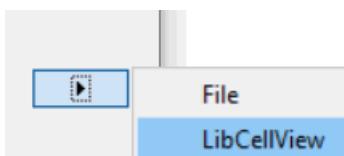
The Select Model dialog box appears.

## Part Developer User Guide

### Creating Verilog Wrappers and Map Files

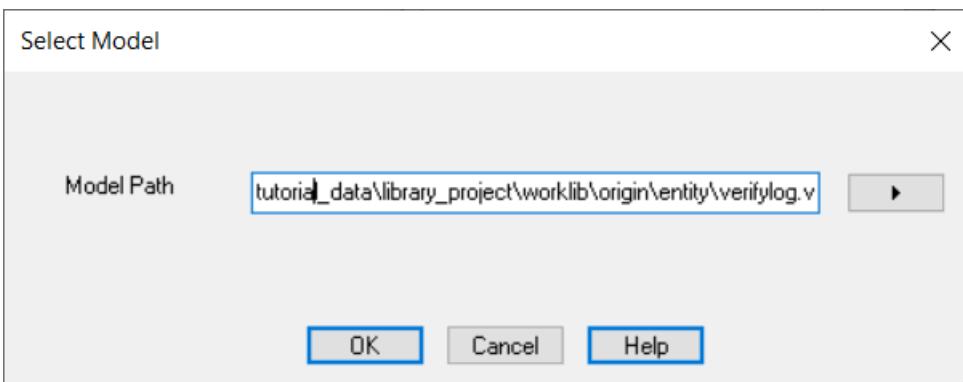


3. Specify the path to the Verilog model in the *Model Path* field. To browse to the Verilog model location, use either the File browser or the Lib:Cell:View browser. The File method is explained here.
4. To launch the file browser or the lib:cell:view option, click on the button with the arrow. The *File/LibCellView* shortcut menu appears.



5. Select *File*.
- The *Open* dialog box appears.
6. Browse to the location where the Verilog model is stored and click Ok.

The *Select Model* dialog box appears with the path to the Verilog model seeded in the *Model Path* field.

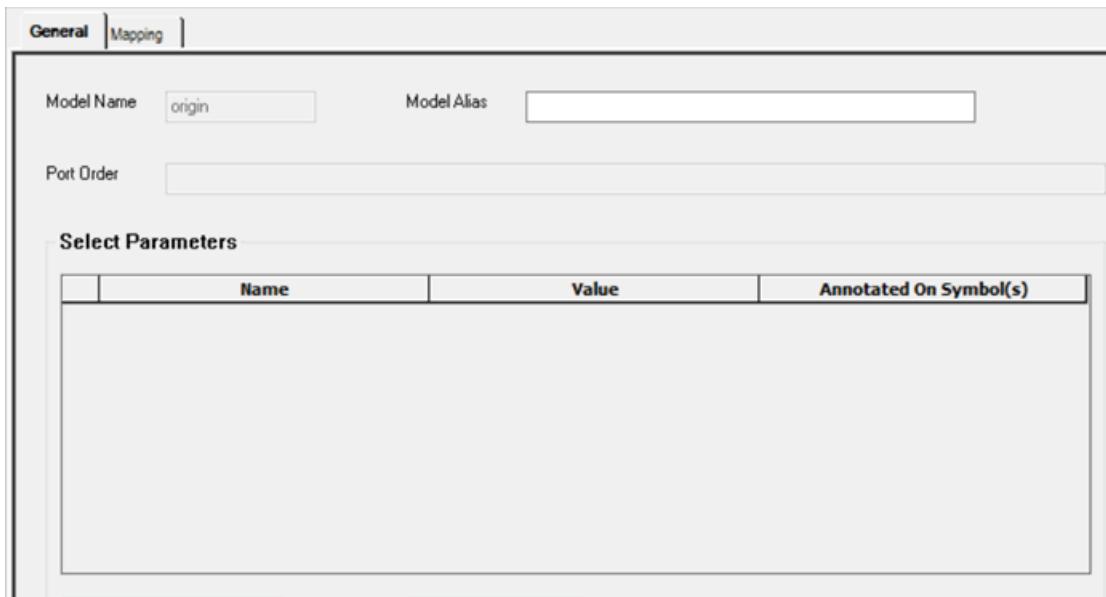


## Part Developer User Guide

### Creating Verilog Wrappers and Map Files

**7.** Click *Ok*.

The *Verilog Map File* editor appears in the Cell Editor.



The *General* page appears in the right pane. There are two non-editable fields on this page, *Model Name* and *Port Order*. The *Model Name* field displays the name of the Verilog model, and the *Port Order* field displays the exact order of ports as they appear in the module.

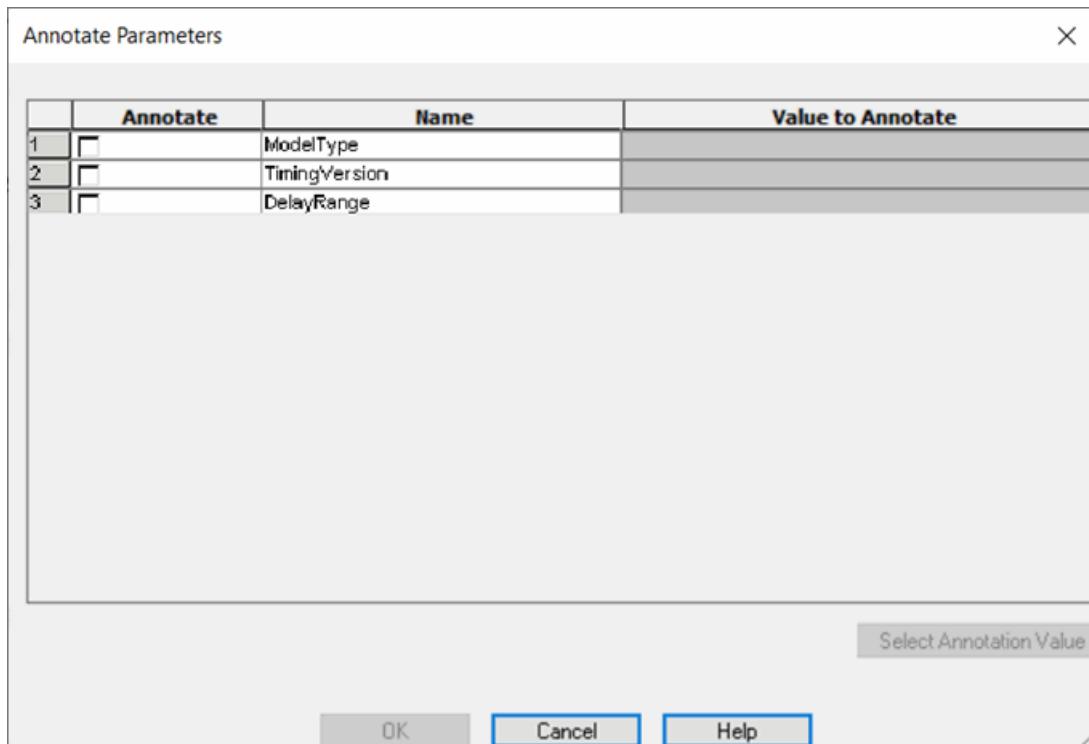
**8.** Specify the aliases for the model in the *Model Class* field.

**9.** The *Select Parameters* group box displays the parameters that exist in the Verilog model. By default, none of these parameters is annotated on the symbols. To annotate a parameter, click *Annotate Parameters*.

The Annotate Parameters dialog box appears. In addition to the parameters present in the selected Verilog model, this dialog box will also show all the parameters that are present in all the Verilog map/wrapper files for the part.

## Part Developer User Guide

### Creating Verilog Wrappers and Map Files



10. Select the parameters that you want to add to the symbols and click *Ok*.

**Note:** When you select a parameter, the value appears in the *Value to Annotate* field. By default, the value is the one that exists for the parameter in the Verilog model. In case you want to change the value, click *Select Annotation Value*. This will display the Select Value to be Annotated dialog box through which you can specify the value for the selected parameter. If a parameter is present in multiple wrappers/map files with different values, all the values are displayed for the parameter through a drop-down list. You can either select one of the existing values from the list or specify a new value.

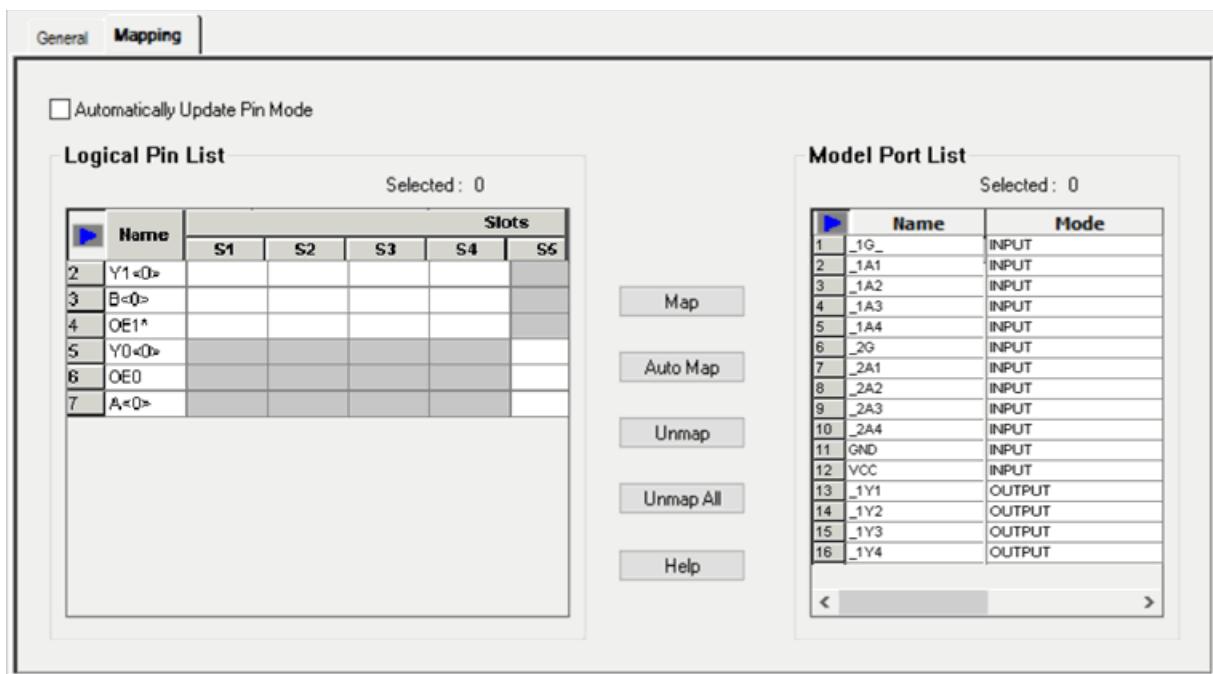
Next, you need to do the mapping of the Verilog model port list with the logical pin list. The mapping is done through the *Mapping* page.

11. Click *Mapping*.

The *Mapping* page appears.

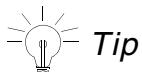
# Part Developer User Guide

## Creating Verilog Wrappers and Map Files



The *Mapping* page displays the logical pin and model port lists. The Logical Pins grid displays the logical pins present in the selected packages, their modes, and the slots in which they are present. The slots for which a pin is absent are disabled. For example, the slot S1 for pin Y0<0> is disabled because the pin is not present in slot S1. Pin mode is determined by the value of the VLOG\_MODE property on the symbol pins. If the property is not found, then the `chips.prt` file is read to determine the pin mode.

### 12. Do the mappings as required.



The *Automap* button provides an automatic way of mapping the model ports to logical pins. See [Auto Map](#) on page 70 for details.

**Note:** The order in which you select the cells or rows will be the order in which the mapping is done. For example, for the logical pin Y1<0>, if you select the slot S2 first and S1 next, and then from the model ports, select \_1Y1 and \_1Y2 in that order, then on mapping, \_1Y1 will get mapped to slot S2 and \_1Y2 will get mapped to slot S1.

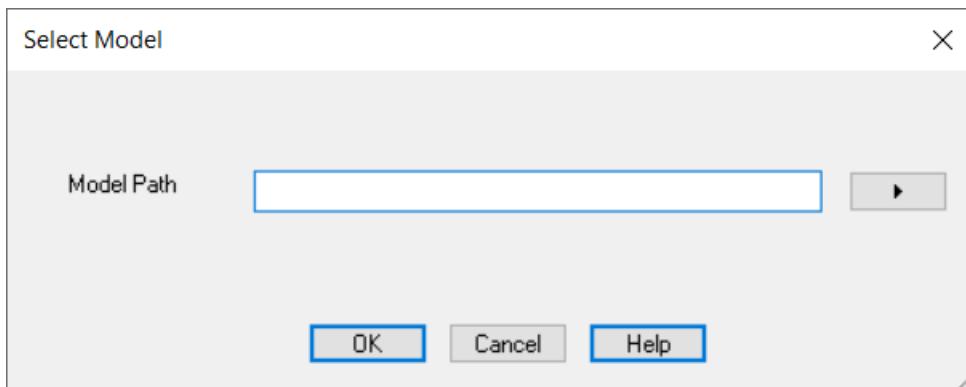
**Note:** In case you are mapping logical pins to model ports of different modes, then you need to check the *Automatically Update Pin Mode* option. Otherwise, Part Developer will give an error and disallow the mapping.

This completes the creation of a Verilog map file.

## Creating a Verilog Wrapper File

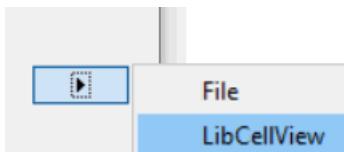
1. Select *File – New – Verilog Wrapper*.

The *Select Model* dialog box appears.



2. Specify the path to the Verilog model in the *Model Path* field. To browse to the Verilog model location, use either the File browser or the Lib:Cell:View browser. The File method has been explained here.
3. To launch the file browser or the lib:cell:view option, click on the button with the arrow.

The *File/LibCellView* shortcut menu appears.



4. Select *File*.

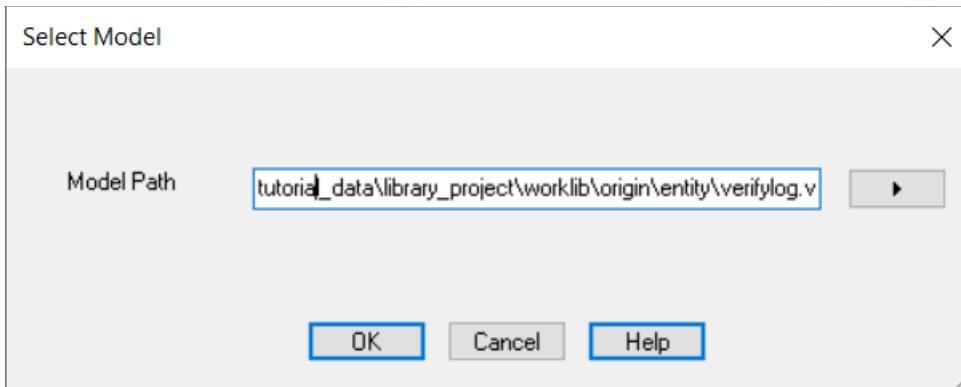
The *Open* dialog box appears.

5. Browse to the location where the Verilog model is stored and click Ok.

The Select Model dialog box appears with the path to the Verilog model seeded in the *Model Path* field.

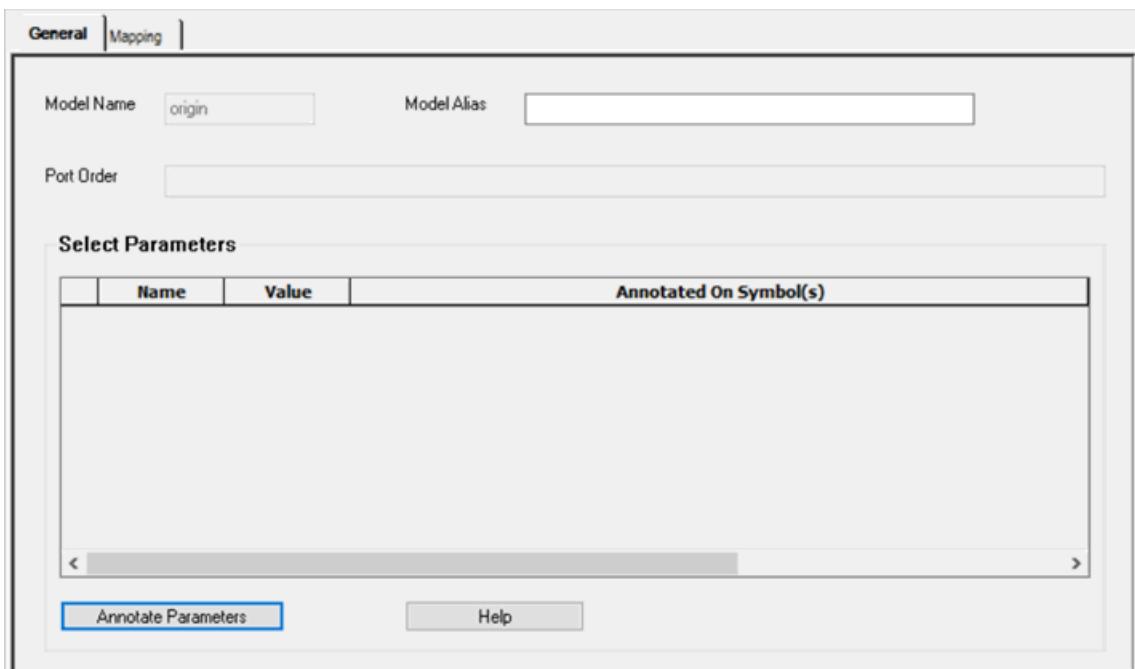
## Part Developer User Guide

### Creating Verilog Wrappers and Map Files



6. Click *Ok*.

The *Verilog Wrapper File* editor appears in the Cell Editor.



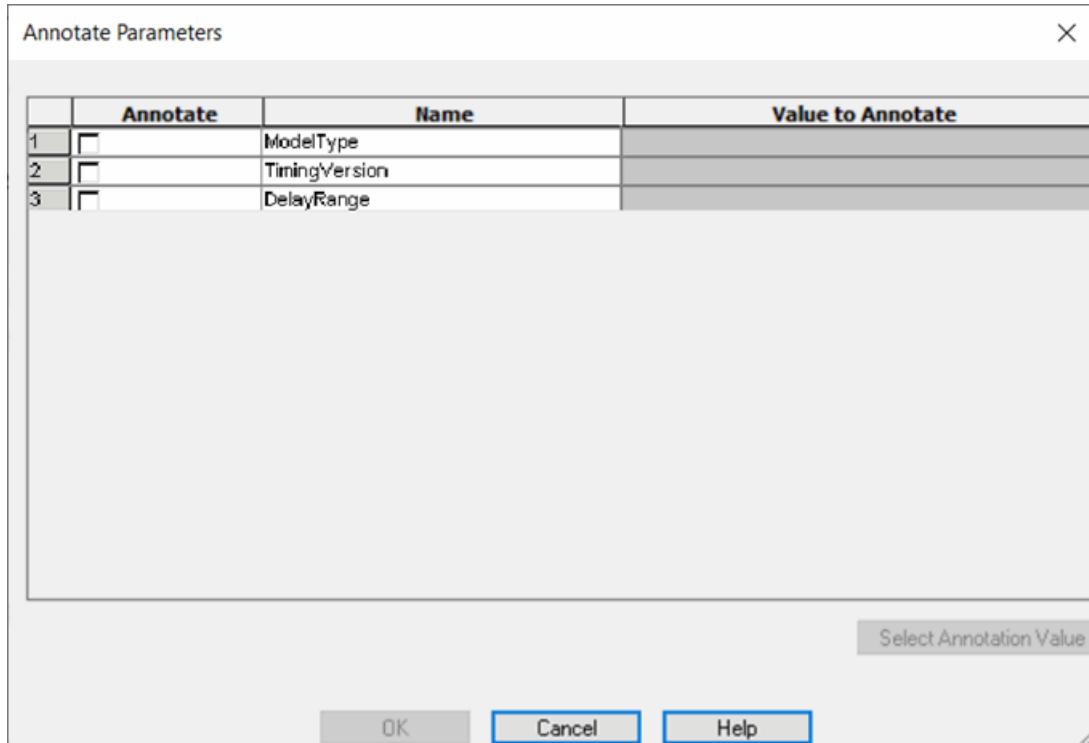
The *General* page appears in the right pane.

7. The *Select Parameters* group box displays the parameters that exist in the Verilog model. By default, none of these parameters is annotated on the symbols. To annotate a parameter, click *Annotate Parameters*.

The Annotate Parameters dialog box appears.

## Part Developer User Guide

### Creating Verilog Wrappers and Map Files



8. Select the parameters that you want to add to the symbols and click *Ok*.

**Note:** When you select a parameter, the value appears in the *Value to Annotate* field. By default, the value is the one that exists for the parameter in the Verilog model. In case you want to change the value, click *Select Annotation Value*. This will display the Select Value to be Annotated dialog box through which you can specify the value for the selected parameter. If a parameter is present in multiple wrappers/map files with different values, then all the values are displayed through a drop-down list. You can either select an existing value or specify a new value.

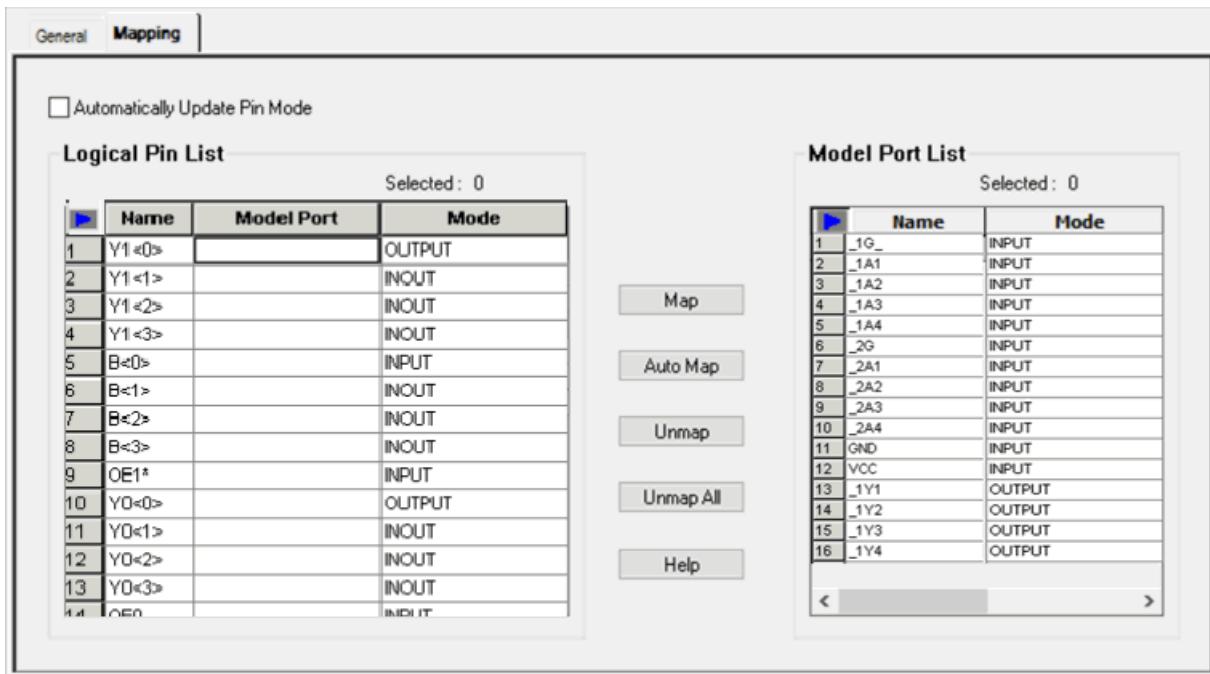
Next, you need to do the mapping of the Verilog model port list with the logical pin list. The mapping is done through the *Mapping* page.

9. Click *Mapping*.

## Part Developer User Guide

### Creating Verilog Wrappers and Map Files

The *Mapping* page appears. This page displays the logical pin and model port lists.



10. To map the logical pin names to Verilog model ports, select the logical pins in the *Logical Pins* list and Verilog model ports in the *Model Port List* and click *Map*.



The *Automap* button provides an automatic way of mapping the model ports to logical pins. See [Auto Map](#) on page 70 for details.

**Note:** The order in which you select the rows will be the order in which the mapping is done. For example, if you select logical pin *Y1<0>* and *Y1<1>* from the *Logical Pins List*, and then select *\_1Y2* and *\_1Y1* in that order from the Verilog *Model Ports List*, then on mapping, *Y1<0>* will get mapped to *\_1Y2* and *Y1<1>* will get mapped to slot *\_1Y1*.

**Note:** In case you are mapping a symbol pin to a model port of different modes, then you need to select the *Automatically Update Pin Mode* option.

This completes the creation of a Verilog wrapper file.

## Modifying a Verilog Wrapper/Map File

After creating wrappers/map files, you may decide to modify the information contained in the wrapper, such as changing the Verilog model or the mapping information etc. Existing wrappers can be modified using Part Developer. To modify a Verilog wrapper:

1. Right-click on the wrapper.
2. Select *Properties*.
3. Make the required changes, such as changing the Verilog model or mappings.
4. Click *OK*.

## Deleting a Verilog Wrapper/Map File

To delete a Verilog wrapper:

1. Select the Verilog wrapper.
2. Press Delete.
3. Click Yes to confirm the deletion.

## Renaming a Verilog Wrapper/Map File

When more than one Verilog wrapper/map file are created, the new wrappers are named `vlog_model[n]`. You may want to change the name to a more intuitive one. Part Developer allows you to rename the Verilog wrappers.

To rename a Verilog wrapper:

1. Right-click on the wrapper to be renamed.
2. Select *Rename*.
3. Modify the wrapper name.
4. Click *OK*.

**Part Developer User Guide**  
Creating Verilog Wrappers and Map Files

---

---

# **Symbol Property Templates**

---

**Note:** This feature is available only with the Allegro Managed Library Authoring licence.

Part Developer provides you the ability to define part construction rules and property settings through the use of templates. Using templates, you can quickly create parts that follow your company standards yet have the flexibility to add properties and behaviors specific to the parts. You can also apply the templates to existing parts.

Typically, a symbol-level template needs to be modularized so that you can create reusable libraries for different symbol elements, such as properties.

To view the information on default Symbol Editor, [\*Symbol Editor User Guide\*](#).

## **Components of a Symbol Property Template**

A symbol property template contains a set of symbol-level properties and their attributes that are added to a symbol when the template is applied. While creating a symbol property template, you can specify the following information:

- Default property height in system units
- Symbol-level properties and their attributes
  - Name
  - Value
  - Visibility
    - Both
    - Invisible
    - Name
    - Value
  - Color

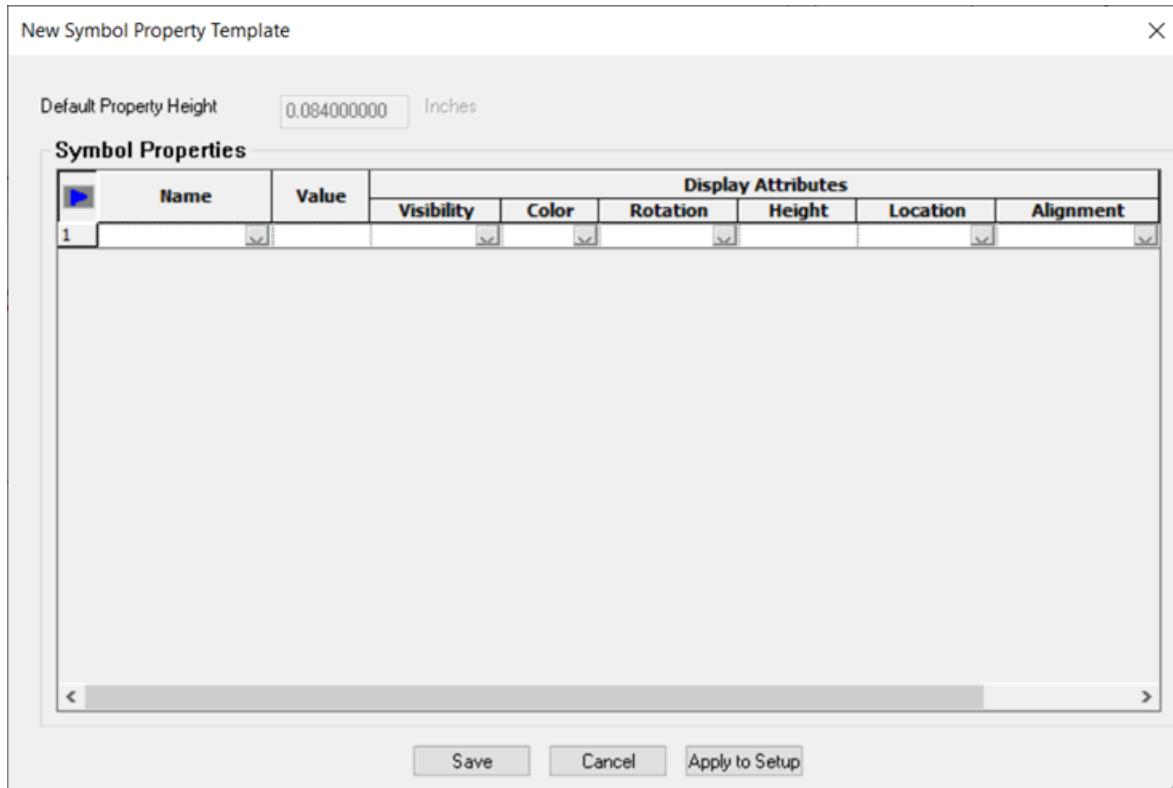
- Rotation
- Height
- Location
- Alignment

## Creating a Symbol Property Template

To create a symbol property template:

1. Choose *Templates – New – Property Template*.

The New Symbol Property Template dialog box appears.



2. Modify the default property height if required.
3. Select a property name from the *Name* drop-down list.  
Alternatively, you can specify the property name in the *Name* field.
4. Specify a property value.

**Note:** If you want the property value to be specified during design entry, you can define the property as a soft property by specifying ? as the property value in the template.

5. Specify the visibility of the property and its value by selecting an appropriate option from the *Visibility* drop-down list.

The following types of visibility are supported in Part Developer:

- Both
- Invisible
- Name
- Value

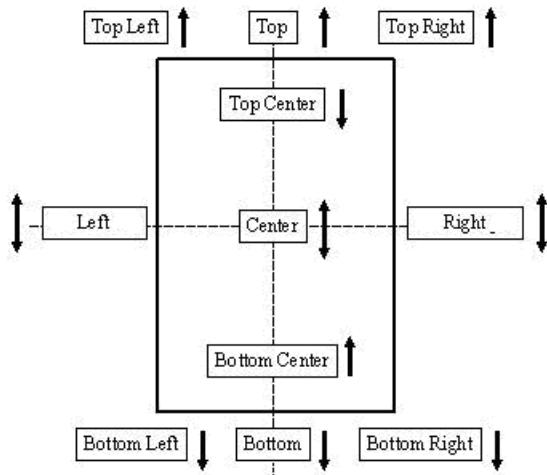
6. Specify the color in which the property and its value are to be displayed by selecting a color from the *Color* drop-down list.
7. Specify the angle at which the property and its value are to be displayed by selecting an appropriate option from the *Rotation* drop-down list.

**Note:** For information about how to configure the rotation of symbol pin properties, see [Configuring the Rotation of Symbol Pin Properties](#) on page 362.

8. Specify the height of the property and its value if the height is required to be different from the default property height.
9. Specify the location at which the property and its value are to be displayed by selecting an appropriate option from the *Location* drop-down list.

**Note:** If more than one property has the same location, Part Developer stacks the

properties in the following way:



 **Important**

If a symbol cannot accommodate all properties with *Center*, *Center - Bottom*, and *Center - Top* locations within the symbol outline, these properties might overlap with properties with adjacent locations.

**Note:** The placement of the first property with each type of location and the space between two adjacent properties with the same location are determined from the values of the following directives in `cds.cpm`:

- `Symbol_Property_Outline_OffsetTop`
- `Symbol_Property_Outline_OffsetLeft`
- `Symbol_Property_Outline_OffsetRight`
- `Symbol_Property_Outline_OffsetBottom`
- `Symbol_Property_Stacking_Offset`

For information on how to configure the symbol property placement and stacking directives, see [Configuring the Placement and Stackup of Symbol Properties](#) on page 363.

10. Specify the alignment of the property and its value by selecting an appropriate option from the *Alignment* drop-down list.

**Note:** For information about how to configure the alignment of symbol pin properties, see [Configuring the Alignment of Symbol Pin Properties](#) on page 362.

11. Click *Save*.
12. Specify a filename.

**Note:** Part Developer saves a symbol property template with a `.sptpl` extension.

13. Click *Save*.

If you want to create parts based on the template in the current session of Part Developer, you can click *Apply to Setup*.

## Opening a Symbol Property Template

You can open an existing symbol property template to modify the template and apply the template to Part Developer setup.

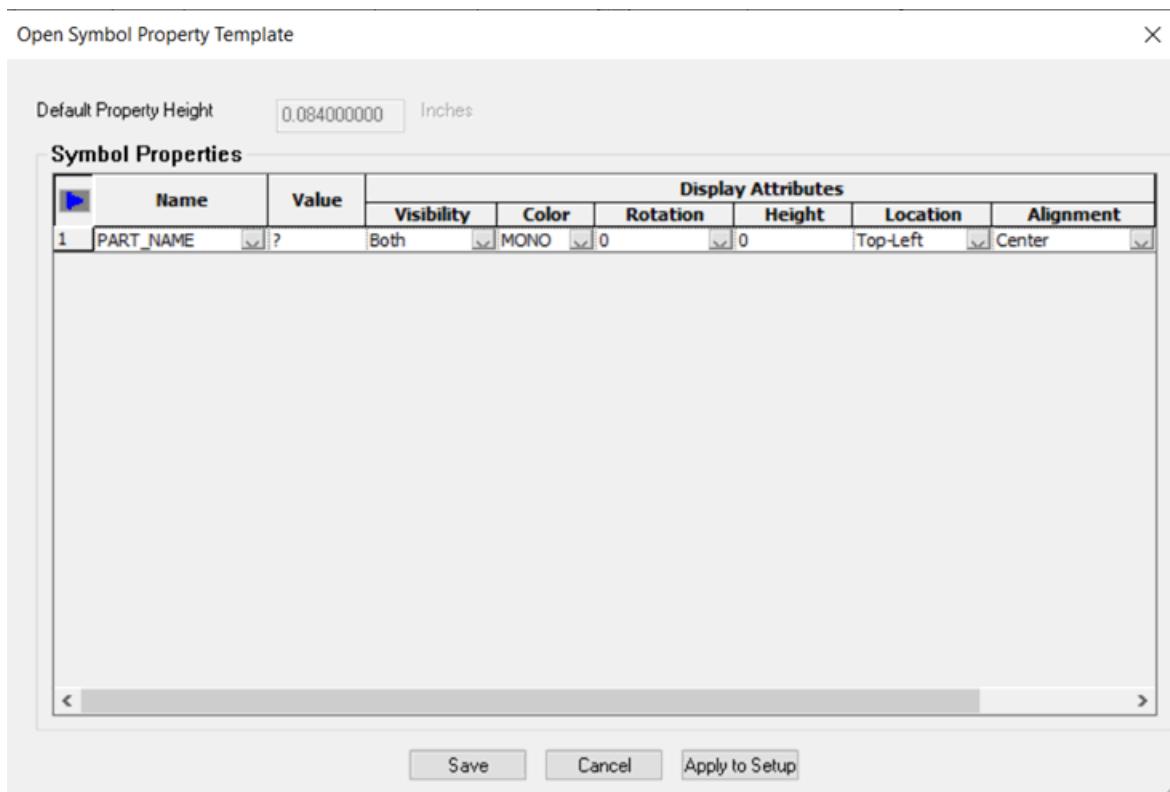
To open an existing template:

1. Choose *Templates – Open – Property Template*.  
The Open dialog box displays.
2. Browse to select the template file.
3. Click *Open*.

## Part Developer User Guide

### Symbol Property Templates

The Open Symbol Property Template dialog box appears.



You can modify the existing properties or add new properties.

#### 4. Click Save.

The modified symbol property template is saved.

If you want to create parts based on the template in the current session of Part Developer, you can click *Apply to Setup*.

## Applying a Symbol Property Template

You can apply a symbol property template in the following ways:

- To a symbol
- To all symbols of a part
- To the setup

After you have applied the template to setup, all parts created in the current session of Part Developer include the properties and values as specified in the applied template.



If the symbol or Part Developer setup contains any symbol property that is not specified in the applied template, the property and its values are retained.

## Applying a Symbol Property Template to a Symbol

To apply a symbol property template to a symbol:

1. Open the part that has the symbol to which the template has to be applied.
2. Right-click on the symbol name in the tree pane.
3. Choose *Apply Property Template*.

The Template Application Wizard appears. The symbol is displayed on the Template Application Wizard canvas.

4. Select the template that you want to load.
5. Click *Open*.
6. Select the *Delete all properties present on the symbol* check box if required.
7. Click *Next*.

The symbol is displayed with the default property height and symbol properties defined in the selected template. You can review the changes and accept or discard them by choosing *Finish* or *Cancel*.

8. Click *Finish* to apply the selected template.

## Applying a Symbol Property Template to All Symbols

To apply a symbol property template to all symbols of a part:

1. Open the part that has the symbols to which the template has to be applied.
2. Right-click on the *Symbols* node in the tree pane.
3. Choose *Apply Property Template To All*.
4. Select the template that you want to load.
5. Click *Open*.

The default property height and symbol properties defined in the selected template are applied to all symbols of the part.

## Applying a Symbol Property Template to the Setup

To apply a symbol property template to the setup:

1. Choose *Templates – Open – Property Template*.  
The Open dialog box displays.
2. Browse to select the template file.
3. Click *Open*.  
The Open Symbol Property Template dialog box appears.
4. Click *Apply to Setup*.

This applies the default property height and symbol properties defined in the selected template to the current session of Part Developer.

## Extracting a Symbol Property Template from an Existing Symbol

If you have a symbol that has been built in compliance with your company standards, you can extract information from it and create a symbol property template.

**Note:** Part Developer retains the location but not the position of extracted symbol properties.

To extract a symbol property template from a symbol:

1. Right-click on the symbol name in the tree pane.
2. Choose *Extract Property Template*.

The Extract Symbol Property Template dialog box displays the default property height and existing symbol properties and their values.

You can save the extracted information as a template or modify the extracted information according to your requirements and then save as a template. You can also apply the symbol property template to the setup.

3. Click *Save*.  
The Save As dialog box opens.

## **Part Developer User Guide**

### Symbol Property Templates

---

- 4.** Specify a filename for the symbol property template.
- 5.** Click *Save*.

This creates a symbol property template with data extracted from the current symbol.

## **Part Developer User Guide**

### Symbol Property Templates

---

## **Import and Export**

---

Part Developer provides the ability to import part information from a variety of input sources, such as EDAXML and Si2PinPak, to create or modify Design Entry HDL and Capture parts, and to export part data in EDA XML and Capture format. This chapter describes the following:

- [Import Export Methodology](#) on page 224
- [Import APD Component Files](#) on page 233
- [Import Capture Part \(Windows Only\)](#) on page 234
- [Import EDAXML Part](#) on page 240
- [Import Si2 PinPak XML Part](#) on page 243
- [Import Comma Separated Value \(.csv\) File](#) on page 246
- [Import Synopsys PTM Model](#) on page 255
- [Import Verilog Model](#) on page 257
- [Import VHDL Model](#) on page 258
- [Import FPGA](#) on page 259
- [Import Text File](#) on page 264
- [Import ViewLogic\(VL\) Part](#) on page 273
- [Import Allegro Footprint](#) on page 277
- [Import Die Text](#) on page 278
- [Import DML Model](#) on page 279
- [Import IBIS Model](#) on page 280
- [Import Mentor Part](#) on page 282
- [Import Pin Grid](#) on page 285
- [Import Pin Table](#) on page 285

## Part Developer User Guide

### Import and Export

---

- [Import ECO - APD Component Files](#) on page 285
- [Import ECO - Capture Part \(Windows Only\)](#) on page 286
- [Import ECO - EDAXML Part](#) on page 287
- [Import ECO - Si2 PinPak XML Part](#) on page 288
- [Import ECO - Comma Separated Value \(.csv\) File](#) on page 289
- [Import ECO - Synopsis PTM](#) on page 289
- [Import ECO - FPGA](#) on page 290
- [Import ECO - Text File](#) on page 291
- [Import ECO - ViewLogic\(VL\) Part](#) on page 292
- [Import ECO - Allegro Footprint](#) on page 293
- [Import ECO - Die Text](#) on page 294
- [Import ECO - DML Model](#) on page 295
- [Import ECO - IBIS Model](#) on page 296
- [Import ECO - Mentor Part](#) on page 297
- [Import ECO - Pin Grid](#) on page 297
- [Import ECO - Pin Table](#) on page 298
- [Export Capture Part \(Windows Only\)](#) on page 299
- [Export EDAXML Part](#) on page 304
- [Export Comma Separated Value \(.csv\) File](#) on page 309
- [Export ViewLogic\(VL\) Part](#) on page 310
- [Export Mentor Part](#) on page 310

## Import Export Methodology

Part Developer enables you to create or modify existing parts using a variety of data sources. This section describes the methodology implemented by Part Developer to create or modify parts using data from the various supported import formats.

## Part Developer User Guide

### Import and Export

---

The import source that is read into Part Developer is a file that will typically have one or more of the following information for one or more parts:

- Cell Name
- Package Name
- Package Aliases
- Package Properties
- Package Pins Name
- Package Pin Width
- Package Pins Type
- Package Pin Mapping
- Package Pin Properties
- Symbol Names
- Symbol Pin Name
- Symbol Pin Width
- Symbol Pin Properties
- Symbol Pin Graphics
- Symbol Pin Location
- Symbol Graphics

It is possible to use these data sources to create a new part or to modify an existing part. The mechanism to modify an existing part by importing part data is called the Engineering Change Order (ECO) process.

### **Methodology Used When Updating Parts Using the ECO Process**

The ECO process is used to modify an existing part by importing data. In addition to selecting the import file format, you need to select the part that is to be modified. Once the part has been selected, Part Developer uses the following mechanism to update the part information:

**Note:** When doing ECO, the values specified in Setup options are used to modify package/symbol information.

## Packages

### ***Package Name and Data***

- If the part being modified has no packages and the package name is not found in the imported data, a package by the name of `newpackage` will be created.
- If the part being modified has only one package, that package is updated with the imported data.
- If multiple packages exist and the package name matches, data is updated for the matched package. If the package name does not match, a new package is created using the imported data.
- The package alias information is taken as is from the imported data. In case the package aliases already exist, the alias names got from the imported data are merged with the existing alias names.
- Package properties are handled in the following way:
  - New property is found in the imported data. In this case, the property is added to the package
  - Property with the same value is found in the imported data. In this case, the property is ignored.
  - Property with a different value is found in the imported data. In this case, the value of property in the package will be overwritten.
- The package pin names are taken as is. In case there are new pins in the imported data, they are added to the package.
- The package pin width information is taken as is from the imported data. In case there already exists a pin with the same pin width, the bus is split on the cell and expanded or contracted as per the imported data.
- In case the pin type information got from the imported data is different from the existing pin type, the pin type got from the imported data is retained.
- The package pin mapping got from the imported data overwrites the existing pin mapping information.
- The package pin properties are handled in the following way:
  - New property is found in the imported data. In this case, the property is added to the package.

## Part Developer User Guide

### Import and Export

---

- Property with same value is found in the imported data. In this case, the property is ignored.
  - Property with a different value is found in the imported data. In this case, the value of property in the package will be overwritten.
- In case of parts where the number of slots in the existing package differs from the number of slots available from the imported data, the larger set is retained. For example, if a part has a package that has four slots and the number of slots in the imported data is only two, then all the slot information of the four slots is retained. The following table describes the behavior of Part Developer when slot information of imported data differs from slot data in the existing package.

Slot information in Imported data	Existing slot information	Action
1	n	n slots are retained
n	1	n slots are retained
N1	N2	If N1>N2, N1 slots are retained and N2 slots are synchronized.  If N2>N1, N2 slots are retained and N1 slots are synchronized.

**Note:** The ECO process works on the assumption that you will re-import the part information to get some more changes. Therefore, the assumption is that the new information is not drastically different from the one imported earlier. The only exception to this case is for split symbols, where you may have imported a flat part and split it to reduce the size of the symbols. This exception is being handled as described earlier. The ECO process tries to import all the information and keep the packages and symbols synchronized. In some cases, it may not be feasible. In such cases, you need to manually synchronize the symbols with the packages.



***By default, Part Developer deletes the properties that are present in the cell but not in the input file. Select the Property deletions option in the Ignore section of the ECO Messages page to turn off this behavior.***

## Symbols

- If the symbol is associated only with the package undergoing ECO, then only that symbol will be modified.
- If ECO is happening in a part with multiple packages, and symbols have multiple associated packages including the package that has to undergo ECO, then in that case symbols are not modified but shown as unassociated with the package that has undergone ECO.

For example, consider a part with three packages, say pkg1, pkg2, and pkg3 and two symbols, sym\_1 associated with pkg1 and pkg2 and sym\_2 associated with pkg3. Now if you do an ECO using import data that does not contain symbol information and pkg1 undergoes ECO, then sym\_1 will not undergo ECO because it is associated with pkg2 as well. Thus, after ECO, sym\_1 will show unassociated with pkg1.

- If the imported data has symbol information, then ECO happens to symbols based on the symbol information. Continuing with the above example, suppose the imported data had information for the three symbols (symbol 1, 2, and 3) as well. In such a case, sym\_1 will undergo ECO with 'symbol 1' information, and sym\_2 with symbol 2.
- Extra pins present in the imported data are added to the symbol based on the setup options.
- Pins that exist both in the symbol and in the imported data, the attributes like pin type and location are modified as per the imported data.

**Note:** There is no deletion of pins from the symbol if the imported data does not have information about pins existing on the symbol.

- The symbol pin width information is taken as is from the imported data. In case there already exists a pin with the same pin width, the bus is split on the cell and expanded or contracted as per the imported data.
- The symbol pin properties are handled in the following way:
  - New property is found in the imported data. In this case, the property is added.
  - Property with the same value is found in the imported data. In this case, the property is ignored.
  - Property with a different value is found in the imported data. In this case, the value of the property in the package will be overwritten.
- The pin shape name that is got from the imported data is used as is if it matches one of the pin shapes supported in Part Developer, else one of the following is done:
  - For new pins, pin\_line is used with the matched length.

- For existing pins, only the length is adjusted.
- Vector pins are added to the symbol in expanded form.
- Only those symbols that have a corresponding entry in the imported data are modified.

### ***Symbol Pin Location***

- If the symbol pin location is not specified in the imported data, then
  - name/type rule matching from the configuration translation rule is used to get the location name for new pins
  - pin location stays unchanged for existing pins
- If the symbol pin location is specified in the imported data, then it matches the location availability on the symbol and overwrites the information in the existing part, else it is treated as if the symbol pin location is not specified.

### **Symbol Graphics**

The symbol graphic is handled in the following way:

- If the symbol locations are the same, then the symbol graphic is replaced with the information from imported data.
- If the symbol locations are different, then a new symbol is created from the imported data.

**Note:** The ECO is designed to re-import the same part information or add part information to the existing part. Depending on case to case, the ECO data can create situations where the part needs to be modified by the user before save. For example, if the part has pin name as XYZ\* and the import data has the same pin defined as XYZ, then the import will create a new pin XYZ. You will need to map it correctly after import.



**Caution**

***By default, Part Developer retains any symbol graphic modifications that have been done during ECO. Select the Graphic modifications option in the Ignore section of the ECO Messages page to turn off this behavior.***

## Example

This example shows how ECO is done on a part with one slot. The csv file that is used for the ECO also has one slot.

The chips.prt file is as follows:

```
primitive 'MULTIPLE_PKG_DIP','MULTIPLE_PKG_DIP-1','MULTIPLE_PKG_CCC';  
  
pin  
'A2':  
  PIN_NUMBER='(1)';  
  PROP1='?';  
  INPUT_LOAD='(-0.01,0.01)';  
  
'B2':  
  PIN_NUMBER='(2)';  
  PROP1='?';  
  OUTPUT_LOAD='(1.0,-1.0)';  
  
'C2':  
  PIN_NUMBER='(3)';  
  PROP1='?';  
  INPUT_LOAD='(-0.01,0.01)';  
  
'D2':  
  PIN_NUMBER='(4)';  
  PROP1='?';  
  INPUT_LOAD='(-0.01,0.01)';  
  
'J'<0>:  
  PIN_NUMBER='(5)';  
  BIDIRECTIONAL='TRUE';  
  INPUT_LOAD='(-0.01,0.01)';  
  OUTPUT_LOAD='(1.0,-1.0)';  
  
'J'<1>:  
  PIN_NUMBER='(6)';  
  BIDIRECTIONAL='TRUE';  
  INPUT_LOAD='(-0.01,0.01)';
```

## Part Developer User Guide

### Import and Export

---

```
    OUTPUT_LOAD='(1.0,-1.0)';
'J'<2>;
    PIN_NUMBER='(7)';
    BIDIRECTIONAL='TRUE';
    INPUT_LOAD='(-0.01,0.01)';
    OUTPUT_LOAD='(1.0,-1.0)';
end_pin;
body
PART_NAME='MULTIPLE_PKG';
JEDEC_TYPE='DIP10_2';
PHYS_DES_PREFIX='U';
CLASS='IC';
end_body;
end_primitive;
```

The csv file has the following entries:

```
package_name,MULTIPLE_PKG_CCC
jedec_type,dip20
assertion_char,_N
Family, CMOS
Who,R,U
Class, IO
pin_number,pin_name,pin_type,prop1,new_prop
1,A_new,TS,test,1
2,B2,INPUT,new,2
14,D2,TS_BIDIR,check,3
6,VCC,POWER
8,Y0*,TS,,5
```

The result of ECO will be as follows:

- The JEDEC\_TYPE property gets modified to dip20.
- The value for the FAMILY package property gets modified to CMOS.
- The pin mappings get updated in the following way:

## Part Developer User Guide

### Import and Export

---

- a. First, pin 1 is searched in the slot S1. It is found as mapped to pin A. This results in pin A being renamed to A\_new. Similarly, other attributes are also synchronized. The pin type is modified from Input to TS and the value of the package pin property Prop1 is modified from ? to TEST. Also, a new property, New\_prop, gets added to the pin.
- b. Next, pin 2 is searched and found mapped to pin B2. Since CSV also has pin B2 mapped to 2, no renaming takes place. However, other attributes are synchronized. For example, the pin type is modified from Output to Input. Similarly, the value of pin property PROP1 is modified from ? to New. Also, a new property, New\_prop, gets added to the pin.
- c. Next, pin 14 is searched and found missing from slot S1. Next, logical pin D2 mapped to 14 in CSV is searched and found mapped to pin 4. This results in the pin number of D2 being modified from 4 to 14. Similarly, other attributes, such as pin type and package pin property value, are also modified and a new property New\_prop gets added to the pin.
- d. Next, pin 6 is searched and found mapped to J<1>. This results in pin 6 getting mapped to power pin VCC.
- e. Next, pin 8 is searched. It is not found in slot S1. So, logical pin Y0\* is searched. Since Y0\* is also missing, this pin gets added as a new pin to the package after ECO and a new property, New\_prop, gets added to the pin.

**Note:** If there was a symbol associated with the package, then the pins renamed in the package get renamed in the symbol as well.

## Import APD Component Files

The *Import APD Component Files* option imports files generated by the `allegro_component` command in APD to create a part.

### Conversion Details

The `allegro_component` command creates a component folder that contains the physical, logical, and pin delay information for a part. Part Developer reads the files in this folder to create the part. If the files have PIN\_DELAY values, these values are written in the new PIN\_DELAY syntax wherever it is required.

For example:

```
'A':  
PIN_NUMBER='(0,0,0,0,1,2,3,4)';  
INPUT_LOAD='(-0.2,0.02)';  
PIN_DELAY='(1:0.2 ns; 2:0.33 ns; 3:0.31 ns; 4:0.31 ns;)';  
'OE0':  
PIN_NUMBER='(0,0,0,0,19,19,19,19)';  
INPUT_LOAD='(-0.2,0.02)';  
PIN_DELAY='(0.2 ns;)';
```

**Note:** In the part information created by the `allegro_component` command, pin names are represented in `<logical_name>_<physical_pin_number>` format. While creating the part, Part Developer splits the format and extracts logical pin names and physical pin numbers. If you want to retain the pin name format of the `allegro_component` command, assign value FALSE to the `Import_APD_strippinnum` directive in the `setup.cpm` file.

### Steps

The steps are as follows:

1. Choose *File – Import and Export*.  
The Import and Export dialog box appears.
2. Choose *Import APD Component Files* and click *Next*.  
The Select Source page appears.

3. Browse to the *component* folder and click *Next*.

The Select Destination page appears.

4. Enter the cell name, select the destination library, and click *Next*.

The Cell Data page appears.

5. If required, you can make changes before the part is created.

6. Click *Finish* to complete the part creation.



**To successfully import APD Component files, the component folder must have at least one report (.rpt) file. These files are read to extract component pin names.**

## Import Capture Part (Windows Only)

The *Import Capture Part (Windows Only)* option converts a Capture symbol into cells with chips and symbol views. All the symbols (Normal and Convert) are translated.

The following types of Capture symbols are translated:

- Normal parts (homogeneous/heterogeneous)
- Library-level symbols (Power/off-page connectors/hierarchical/title)

**Note:** The attached implementation is not translated by the tool.



**If a Capture part has a pin number as 0, the pin gets converted to pin number zero in a Design Entry HDL part. This is because Design Entry HDL does not support 0 as a pin number.**



**If a Capture Part has some of the pin numbers missing, then it will create an incomplete chips file due to missing data in the PIN\_NUMBER property. You need to manually edit such a chips file to make it work with Design Entry HDL. For example, in the 74LS30 part covered later in the examples section, notice that pin numbers 7, 9, and 10 are missing. If you were to convert such a part, then the resulting chip file will not have the**

***pin numbers 7, 9, and 10. To ensure that they appear in the chips file, you should add a user property called NC and add the pin numbers 7, 9, and 10 as its value. This causes the pins 7, 9, and 10 to appear as NC pins in the chips file.***

## Conversion Details

### Conversion of the Chips (Physical) View

The following information is translated from the Capture part:

- Pin names
- Pin numbers
- Package properties
- Pin types
- Part aliases

#### ***Pin Names***

- The pin names are translated into Design Entry HDL as pin names in the chips view and as pin text on the symbol pins in the symbol view.

Due to restrictions in Design Entry HDL on certain characters, the following characters are replaced:

- > is replaced with \_GT\_
- < is replaced with \_LT\_
- ‘ is replaced with \_

- In the case of duplicate pin names:

- If all the pins are visible and are of type POWER, the pin is created as a vector pin in Design Entry HDL.
- If all pins are not of type POWER, then all non-POWER pins that have duplicate pin names are converted to unique pin names in Design Entry HDL by appending \_<number> suffix to the pin name. The numbering starts from 1. For example, if the Capture part had 5 pins with the name A, they will be converted to A\_1, A\_2, A\_3,A\_4, A\_5 pins in Design Entry HDL.

- ❑ If all the pins are invisible and are of type POWER, all the pins are written in the body section of the chips file in Design Entry HDL.
- ❑ If a part has duplicate pins of type POWER and some pins are visible and some are not, then for visible pins, \_<number> is appended and the invisible pins are moved to the body section in the chips view in Design Entry HDL.
- The low assertion character (\* or \_N) in the pin names is set on the basis of the settings in the Design Entry HDL project file.

### **Pin Numbers**

- If the Capture part has no physical pin numbers, no chips view is created in Design Entry HDL.

**Note:** For additional information on pin numbers, see the *Caution* section above.

### **Package Properties**

The properties translated are as follows:

- PCB footprint to JEDEC\_TYPE
- Part Reference Prefix to PHYS\_DES\_PREFIX
- Name to Cell name and PART\_NAME

**Note:** A Design Entry HDL part has the capability to support additional data on pins, such as pin loads. This information is not present in Capture and is therefore seeded with the values set in the project settings. The SIZE property is not transferred as it has no relevance in the Design Entry HDL domain.

### **Pin Type**

The pin types are converted as per the table listed below:

Capture Pin Types	Design Entry HDL Pin Type
INPUT	INPUT
OUTPUT	OUTPUT
Open Collector	OC

## Part Developer User Guide

### Import and Export

---

Capture Pin Types	Design Entry HDL Pin Type
Open Emitter	OE
3-State	TS
Bidirectional	BIDIR
POWER	POWER
PASSIVE	ANALOG

### ***Part Alias***

As discussed in the steps, you have the choice to retain or drop the Capture part aliases. If aliases are to be retained, you have a choice to create them as packages in the same chips view or create them as different cells.

### **Conversion of Symbol View (Logical)**

The following symbol information is translated from a Capture symbol:

- Pin Names
- Pin Location
- Graphics - Symbol Shape
- Graphics - Pin Shapes
- Pin Properties
- Symbol Properties

### ***Pin Names***

The pin names are matched to the names in the chips view.

### ***Pin Location***

Special care is taken to ensure that the location of pins is an exact match to that in the Capture part. This allows the design translators to be based on graphical translation. Refer to the *Caution* section if you find a pin missing.

### ***Graphics - Symbol Shapes***

The following graphics entities are translated:

- Line
- Rectangle
- Ellipse
- Arc
- Polyline
- Filled shapes

**Note:** The bitmaps are not translated as they are not supported in Design Entry HDL. The filled shapes are shown as unfilled in Design Entry HDL as Design Entry HDL does not support filled shapes.

**Note:** The fonts in a Capture symbol are not translated as Design Entry HDL does not support fonts.

### ***Graphics - Pin Shapes***

The Capture pin shapes are translated into Design Entry HDL to their equivalent graphics.

### ***Pin Properties***

The following properties are not transferred from Capture symbol ports:

- Name
- Number
- Order
- Long
- Clock
- Dot
- Pin\_Number\_Is\_Visible

### **Symbol Properties**

- If the property has no value in Capture, it is not transferred to the converted Design Entry HDL part.
- Each space in a property is converted to an ‘\_’.
- The following property names and case are modified:
  - Implementation Path to IMPLEMENTATION\_PATH
  - Implementation Type to IMPLEMENTATION\_TYPE
  - Implementation to IMPLEMENTATION
  - PSpiceTemplate to PSPICETEMPLATE
  - NC to NC\_PINS (this symbol property is transferred to the chips view)
- The following properties are not added to the Design Entry HDL symbol from the Capture symbol:
  - Name
  - Value
  - Part\_Reference
  - Reference

In both the above-mentioned cases, it is recommended that the Capture part is corrected before proceeding with translation.

### **Steps**

The steps are as follows:

1. Choose *File – Import and Export*.  
The Import and Export dialog box appears.
2. Choose *Import Capture Part (Windows Only)* and click *Next*.  
The Select Source page appears.
3. Specify the Capture library and click *Next*.  
The Select Capture Part page appears.
4. Select the Capture part from the *Part* drop-down list box.

**Note:** If the selected part has aliases, they get displayed in the *Aliases* list box. By default, only the master components are shown in the *Part* drop-down list box. Therefore, in case you do not find a part listed in the drop-down list box, you need to find its master component and then convert the part.

5. Select whether you want to convert only the master component, each alias as an individual primitive, or each alias as an individual cell.

If you choose to convert only the master component, then one Design Entry HDL part is created. The `chips.prt` file will have only one primitive section with the master component name as its only entry.

If you choose to convert each alias as individual primitives, then one Design Entry HDL part is created with each alias appearing as a primitive entry in the `chips.prt` file. For example, if a part has four aliases, the resultant `chips.prt` file will have four primitives.

If you choose to convert each alias as an individual cell, then as many number of parts as there are aliases will be created. For example, if a part has four aliases, then four separate Design Entry HDL parts will be created.

6. Click *Next*.

The Select Destination page appears.

7. Select the library in which you want to save the converted part and click *Finish*.

The part is created.

## Importing a Capture Library

If you want to convert an entire Capture library into a Design Entry HDL library, you can use the `cap2con` command. For more information on the `cap2con` command, see [cap2con](#) on page 386.

## Import EDAXML Part

Part Developer enables you to create parts from the E-tools XML documents.

**Note:** The setup values are applied to the imported part.

The steps to create a part from an XML datasheet are:

1. Choose *File – Import and Export*.

The Import and Export wizard appears.

**2.** Choose *Import EDAXML Part* and click *Next*.

The Select Source page appears.

**3.** Specify the XML file and click *Next*.

The Select Destination page appears.

**4.** Decide on whether you want to convert only the master component, each alias as an individual primitive, or each alias as an individual cell.

If you choose to convert only the master component, then one Design Entry HDL part is created. The `chips.prt` file will have only one primitive section with the master component name as its only entry.

If you choose to convert each alias as individual primitives, then one Design Entry HDL part is created with each alias appearing as a primitive entry in the `chips.prt` file. For example, if a part has four aliases, the resultant `chips.prt` file will have four primitives.

If you choose to convert each alias as an individual cell, then as many number of parts as there are aliases will be created. For example, if a part has four aliases, then four separate Design Entry HDL parts will be created.

**5.** Select the Design Entry HDL library where you want to store the part and click *Finish*.

## Post Import Issues

After you read in the part information from the XML datasheet, you need to take care of the following issues:

- [PIN\\_TYPE](#) Property on page 242
- [CLASS](#) Property on page 242
- [Duplicate Pins](#) on page 242
- [Pins With '>' or '<'](#) on page 242
- [Partnames with '/'](#) on page 242
- [JEDEC\\_TYPE](#) on page 243
- [NC Pins](#) on page 243
- [Symbol Interpretation](#) on page 243

## **PIN\_TYPE Property**

The PIN\_TYPE property gets filled in as per the following rules:

- Passive is interpreted as UNSPEC.
- The rest of the pins are mapped to their corresponding types in Part Developer.

## **CLASS Property**

The CLASS property gets filled in as per the following rules:

- CLASS property gets the value IC if the Designator element in the XML datasheet had the value U.
- CLASS property gets the value IO if the Designator element in the XML datasheet had the value J, JP, or SW.
- CLASS property gets the value DISCRETE for all other values of the Designator element type. These values are C, D, ISO, L, LS, Q, R, T, U, VR, and Y.

Therefore, you may need to check the value of the CLASS property after the data has been imported from the XML datasheet. If required, change the value of the CLASS property.

## **Duplicate Pins**

Often, a part may have duplicate pin names, such as multiple collector pins in transistors and programmable IO pins in FPGAs. In such cases, Part Developer appends an index with ‘\_’ to the pin name starting from the second occurrence. For example, if there are two collectors with name C, they will be read in as C and C\_1.

## **Pins With ‘>’ or ‘<’**

Certain pins have either a ‘>’ or a ‘<’ symbol in their pin names, such as P>Q or P<Q in comparators. Such pins will be read in with the ‘>’ symbol converted to ‘\_GT\_’ and the ‘<’ symbol converted to ‘\_LT\_’ in the pin names. For example, a pin name P>Q will get converted to P\_GT\_Q.

## **Partnames with ‘/’**

Some parts may have ‘/’ symbol in their part names. The ‘/’ in such part names will get converted to ‘\_’ in the primitive entry of the `chips.prt` file.

## **JEDEC\_TYPE**

When a part information is imported from an XML datasheet, the JEDEC\_TYPE entry is not seeded. You will need to manually specify the value of the JEDEC\_TYPE property for a part.

## **NC Pins**

E-tools XML datasheets do not contain information about the NC pins. Therefore, when you import the data from the XML datasheets, NC pins would be absent from the packages.

## **Symbol Interpretation**

After you create a part from an XML datasheet, you need to be aware of the following:

- The ORIGIN of the symbol will be at the top-left corner of the symbol bounding box, and the PART\_NAME will be displayed just above the ORIGIN.
- Pin text will appear outside the symbol graphic.
- In data sheets, the multi-assertion pin names appears as name/name. When such pins are read into Part Developer from XML, each character in the low-asserted part of the name is followed by a '\'. For example, RD/WR will be written as RD/W\R\.
- Sometimes, some of the symbols may appear to have additional height. This is because when a part data is imported from XML, all the power pins are moved to the body section of the `chips.prt` file. This results in the removal of power pins from the symbols. However, the space reserved by the symbol for power pins is not adjusted after the power pins are moved to the body section.

## **Import Si2 PinPak XML Part**

Si2 PinPak XML format is a detailed model for unambiguous exchange of electronic component (EC) pin map and package information. For more information about the Si2 PinPak specification, see:

[www.si2.org/pinpak/](http://www.si2.org/pinpak/)

## **Conversion Details**

When using PinPak XML file as data source, the part gets created with one package. The package is created with the information based on the pins present in the `Pinmap` section of the XML file.

## Part Developer User Guide

### Import and Export

---

- The `pid` and `symLab` attributes of the `Product` tag and the `shape` attribute of the `Pinmap` tag are added as package properties. The property names are `PID`, `SYMLAB`, and `SHAPE`. The values of these properties are same as that of the attribute value in the XML file.
- The `PinSet` tag is used to derive the vector pins in the following way:
  - The `<Order = bus>` attribute determines that it is a vector pin.
  - The `members` attribute is used to derive the pin IDs for all the pins that form the vector pin.
  - The `label` attribute is used to derive the pin names.
  - The `lsb` attribute is used to derive the least significant bit.
  - The `msb` attribute is used to derive the most significant bit.
- The attributes of the `Pin` tag is used to get the pin information in the following way:
  - The `name` attribute is used to derive the pin name. If the presence of the character `*` in a pin with negative polarity results in an invalid pin name, then Part Developer replaces the `*` in the pin name with `_STAR_`.
  - The `num` attribute is used to derive the pin number.
  - The `pol` attribute is used to derive the pin assertion. If the attribute value is negative, then the pin is created as low-asserted. The pin is considered high-asserted for all other values.
  - The `desc` attribute is added as the `DESCRIPTION` package property with the value of the attribute as the property value.
  - The combination of `type` and `dir` attributes is used to determine the pin type in the following way:

---

Type/ Direction	Input (I)	Output (O)	BIDIR (B)	Undefined (U)
DATA	INPUT	OUTPUT	BIDIR	UNSPEC
ADDRESS	INPUT	OUTPUT	BIDIR	UNSPEC
POWER	POWER	POWER	POWER	POWER
GROUND	GROUND	GROUND	GROUND	GROUND
UNCONNECTED	INPUT	OUTPUT	BIDIR	UNSPEC

## Part Developer User Guide

### Import and Export

---

Type/ Direction	Input (I)	Output (O)	BIDIR (B)	Undefined (U)
ANODE	INPUT	OUTPUT	BIDIR	UNSPEC
BASE	INPUT	OUTPUT	BIDIR	UNSPEC
CATHODE	INPUT	OUTPUT	BIDIR	UNSPEC
CLOCK	INPUT	OUTPUT	BIDIR	UNSPEC
COLLECTOR	INPUT	OUTPUT	BIDIR	UNSPEC
CONTROL	INPUT	OUTPUT	BIDIR	UNSPEC
DRAIN	INPUT	OUTPUT	BIDIR	UNSPEC
EMITTER	INPUT	OUTPUT	BIDIR	UNSPEC
ENABLE	INPUT	OUTPUT	BIDIR	UNSPEC
GATE	INPUT	OUTPUT	BIDIR	UNSPEC
SOURCE	INPUT	OUTPUT	BIDIR	UNSPEC
UNDEFINED	INPUT	OUTPUT	BIDIR	UNSPEC
UNPOPULATED	INPUT	OUTPUT	BIDIR	UNSPEC

## Steps

The steps are as follows:

1. Choose *File – Import and Export*.

The Import and Export wizard appears.

2. Choose *Import Si2 PinPak XML Part* and click *Next*.

The Select Source page appears.

3. Browse and select the input Si2 PinPak file.

The Select Destination page appears.

4. The name of the part to be created is seeded automatically. If required, change the part name.

5. Select the library in which the part is to be created and click on *Finish*.

The Cell Editor appears with the part information.

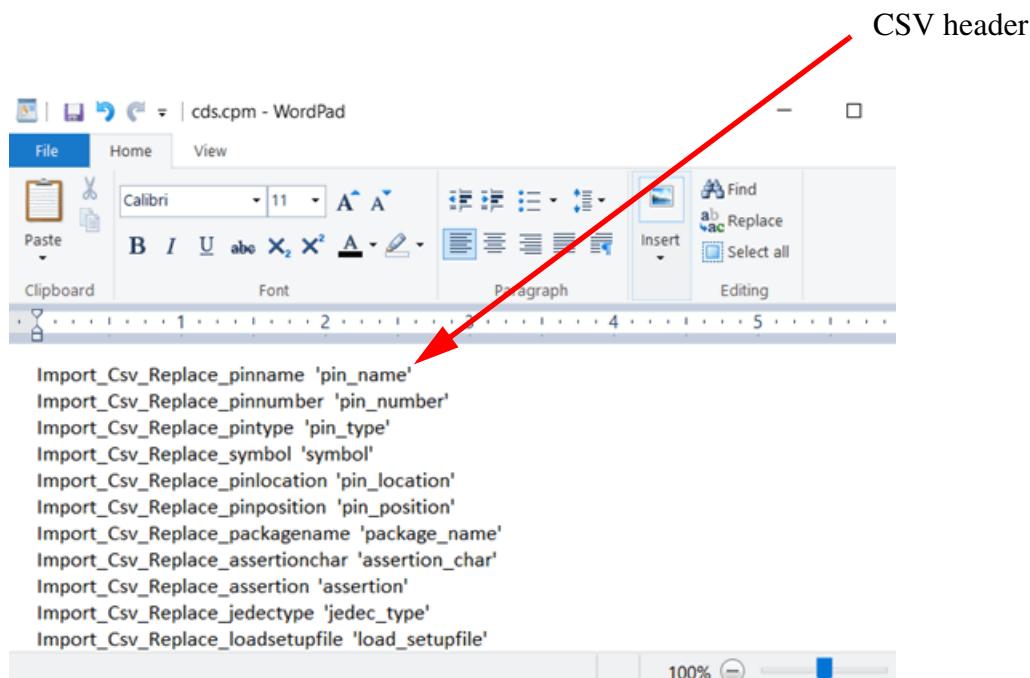
## Import Comma Separated Value (.csv) File

Part Developer can import part information stored in a comma-separated value (.csv) file and create packages and symbols from it.

### Conversion Details

The data in the CSV file should be stored in the name-value pair format for package and pin properties.

By default, a set of key package and pin property names are predefined in `cds.cpm` to indicate how Part Developer should import CSV data to create different information about a part.



If the CSV file contains information about properties that are not predefined in `cds.cpm`, Part Developer imports the information as is in the case of user-defined properties. In the case of standard Cadence properties, Part Developer imports the information only if the values are specified in the correct syntax. For example, the `ALT_SYMBOLS` property, which is a standard Cadence property but does not appear in the list of default properties in CSV import, is imported if its values are specified in the following syntax:

```
(Subclass:Symbol ,... ; Subclass:Symbol ,...)
```

For more information on standard Cadence properties, see *Allegro Platform Properties Reference*.

#### *Important*

If required, you can modify the `cds.cpm` file at `<CDS_SITE>/share/cdssetup/projmgr` to rename the predefined properties according to your requirements. To learn more about the procedure, see [Configuring the Predefined Headers for CSV Import](#) on page 357.

### Predefined Package Properties

Each package property name and its value must be specified as a comma-separated list in a row of the CSV file.

The following package property names are predefined in `cds.cpm`:

- `package_name`
- `jedec_type`
- `assertion_char`
- `load_setupfile`

The package information is determined in the following way:

- The `package_name` property definition is used to derive the name of the package. If this entry is missing, the cell name is used as the package name. You can specify equivalent package names by specifying comma-separated values.

For example, you can specify:

```
package_name,7400_DIP,7400_CCC
```

- The `jedec_type` entry is used to determine the value of the `JEDEC_TYPE` property.
- The `assertion_char` entry is used to determine which pins will be treated as low asserted. If this entry is present, the *Read/Write* field value specified in *Setup* is ignored.
- The `load_setupfile` entry is used to determine the location of the project file from which to read load setup values. If `load_setupfile` is specified, the load setup values of the current project are ignored for the part.

## Predefined Pin Properties

Pin property names and their values for each pin are specified as comma-separated lists in different rows of the CSV file. For example:

```
pin_name,pin_number,pin_type,pin_location,symbol,pin_position,pin_shape,pin_delay  
INPUT,1,INPUT,right,1,4,line_arrow_in,2  
VCC,13,POWER,,,,
```

**Note:** The number of commas in each row of pin definition should match the number of commas in the header row.

The following pin property names are predefined in `cds.cpm`:

- `pin_name`
- `pin_number`
- `pin_type`
- `assertion`
- `pin_location`
- `pin_position`
- `symbol`
- `pin_shape`
- `diff_pair_pins_pos`
- `diff_pair_pins_neg`



The following headers are mandatory:

- `pin_name` and `pin_number` for flat parts
- `pin_name`, `pin_number`, and `symbol` for multi-section parts

The package and symbol information is determined in the following way:

- Entries under the `pin_name` column are read as pin names.
- Entries under the `pin_number` column are read as pin numbers.
- Entries under the `pin_type` column are read as pin types.

## Part Developer User Guide

### Import and Export

- The assertion entry is used to determine whether a pin is low-asserted or not. The strings that determine low assertion are specified through the Import\_Csv\_LowassertFlag directive. By default, the values L and Low are specified.
- Entries under the pin\_location column are used to determine the pin location for specific pin types.
- Entries under the pin\_position column determine the position of the pin from the origin. This will appear as the value of the *Position* property in the Symbol Editor.
- Entries under the symbol column are used to create slots and symbols. Symbols are created only if pin\_location and pin\_position values are also present.

**Note:** If no symbol information is present for POWER, NC, or GND pins, these pins are automatically moved to the *Global Pins* grid. In such a case, a pin number such as <1,2,3,4> is treated as four physical pins.

**Note:** If a pin exists on multiple symbols with different pin types, one of the pin types is imported and the rest is ignored. If this data loss is to be avoided, you must correct the CSV before starting import.

**Note:** If the pin type for a pin is different across slots, Part Developer converts all the pin types to the pin type of the pin in the first slot. This is done because Cadence flows do not support different pin types across different slots for the same pin.

- Entries under the pin\_shape column are used to attach pin shapes from the shape libraries specified in Setup.

#### *Important*

The shape information for a pin is used only when the symbol information is also available.

- Entries in the diff\_pair\_pins\_pos and diff\_pair\_pins\_neg columns are used to derive differential pair information. Two pins having the same diff\_pair\_pins\_pos and diff\_pair\_pins\_neg column values constitute a differential pair, which is identified by the column value.

PIN_NUM	PIN_NAME	PIN_TYPE	SYMBOL	PIN_LOC	DIFF_PAIR_PINS_POS	DIFF_PAIR_PINS_NEG
7	1 A1+	ANALOG		1 Left	a1	
8	21 A1-	ANALOG		1 Left		a1
9	2 A2+	ANALOG		1 Left	a2	
10	22 A2-	ANALOG		1 Left		a2
11	3 A3+	ANALOG		1 Left	a3	

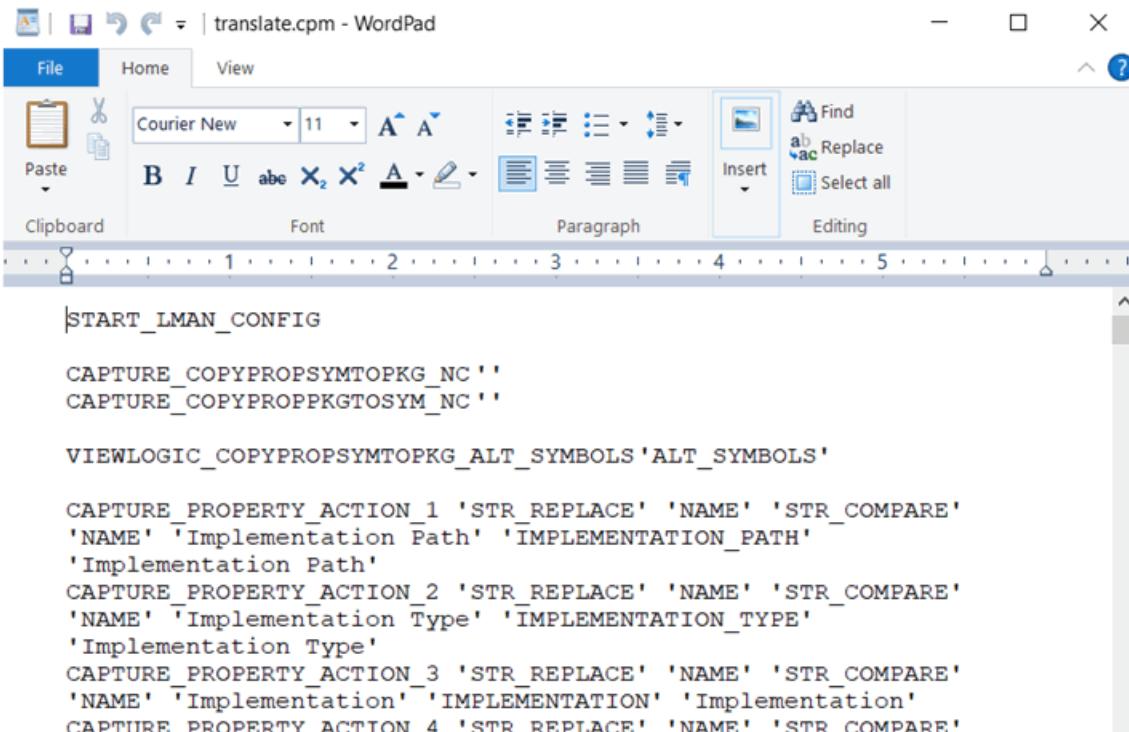
## User-Defined Pin Properties

The following pin properties are also imported:

- PIN\_GROUP
- PIN\_DELAY

## Error Handling

Invalid characters in pin names are automatically converted to supported pin names through the entries provided in the `translate.cpm` file, which is located at `<your_inst_dir>/share/cdssetup/lman`.



A screenshot of the Microsoft WordPad application window titled "translate.cpm - WordPad". The window shows a Microsoft Word ribbon interface with tabs like File, Home, View, Insert, and Editing. The main content area displays a series of text commands in Courier New font, likely representing a script or configuration file. The text includes commands for capturing properties, defining symbols, and performing string replacements.

```
START_LMAN_CONFIG

CAPTURE_COPYPROPSYMTOPKG_NC ''
CAPTURE_COPYPROPPKGOTOSYM_NC ''

VIEWLOGIC_COPYPROPSYMTOPKG_ALT_SYMBOLS 'ALT_SYMBOLS'

CAPTURE_PROPERTY_ACTION_1 'STR_REPLACE' 'NAME' 'STR_COMPARE'
'NAME' 'Implementation Path' 'IMPLEMENTATION_PATH'
'Implementation Path'
CAPTURE_PROPERTY_ACTION_2 'STR_REPLACE' 'NAME' 'STR_COMPARE'
'NAME' 'Implementation Type' 'IMPLEMENTATION_TYPE'
'Implementation Type'
CAPTURE_PROPERTY_ACTION_3 'STR_REPLACE' 'NAME' 'STR_COMPARE'
'NAME' 'Implementation' 'IMPLEMENTATION' 'Implementation'
CAPTURE_PROPERTY_ACTION_4 'STR_REPLACE' 'NAME' 'STR_COMPARE'
```

For example, because ! is not a supported character in Cadence flow, it is translated to \_EXCL\_ when importing the `example.csv` file. In addition, the fifth pin number specified in the CSV file for the BUS vector pin is automatically ignored because the vector pin has only four bits.

## Duplicate Pin Name Handling

The Duplicate Pin Resolver dialog box enables you to handle duplicate pin names. You can choose to make the duplicate pins as bits of a vector pin, individual scalar pins, or move them to the global pin list. For example, consider two pins with name A. If you choose Vector, then the pins are created as A<1> and A<2>. If you select Scalar, the pins are created as A1 and A2.

**Note:** In case scalar or vector pins already exist in the CSV file, the duplicate pin names are automatically incremented to the next available highest integer. For example, if pins A<1> and A<2> already exist, the duplicate pins are created as A<3> and A<4>.

## CSV Import Example

Consider the CSV file `example.csv`, which has the following entries:

These properties are imported as package properties and can be viewed on the General page.

The assertion\_char value is used in place of the Read/Write field value.

	A	B	C	D	E	F	G	H	I
1	part_num	XC2V60000-5FF11521							
2	pack_type	2v6000000ff1152							
3	package_r	consolidated_dip							
4	jedec_type	dip_20							
5	alt_symbol	(dip20_3)							
6	assertion_#								
7	pin_name	pin_number	pin_type	pin_location	symbol	pin_position	pin_shape		
8	INPUT	1	INPUT	right	1	4	line_arrow_in		
9	OUTPUT	2	OUTPUT	top	1	2	line_arrow_out		
10	OC#	3	OC	centre	1	7	line_arrow_in_low		
11	OE	4	OE	left	1	8	line_arrow_in		
12	OC_BIDIR	5	OC_BIDIR	left	2	9	line_bidir_arrow_left		
13	BUS<3..0><6,7,8,9,10>	BIDIR		right	2		line_bidir_arrow		
14	VDD<0>	11	POWER		3				
15	VDD<1>	12	POWER		3				
16	VCC	13	POWER						
17	VCC	14	POWER						
18	GND	15	GROUND						
19	GND	16	GROUND						
20	NC	17	NC						
21	NC	18	NC						
22	NC	19	NC						
23	INVALID!	20	INPUT						

These properties are imported as pin properties and can be viewed on Package Pin and Symbol Pins pages. Their values for each pin are displayed in the rows be-

**Note:** The CSV file has been opened in a spreadsheet viewer to enhance readability.

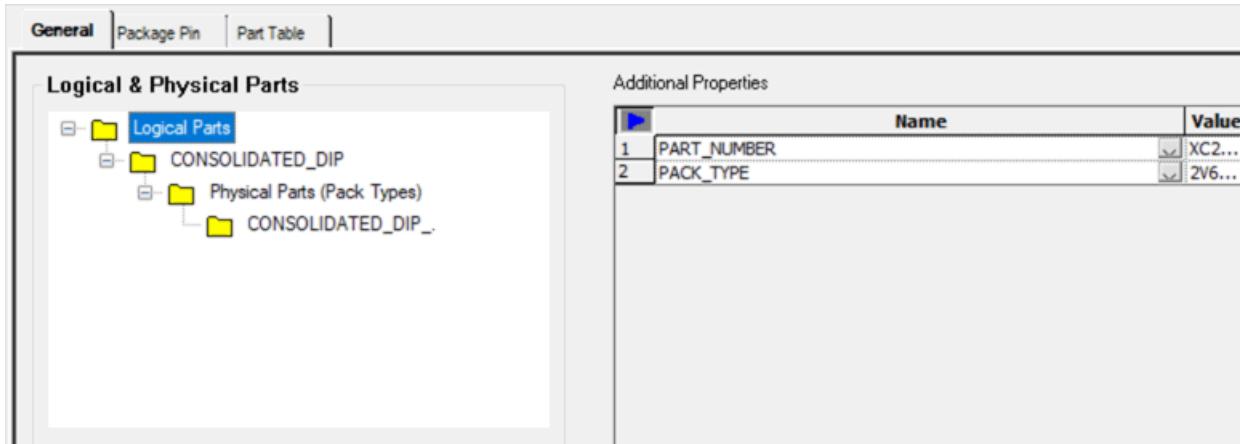
When the `example.csv` file is imported in Part Developer, a part called `example` is created with a package called `CONSOLIDATED_DIP`. The part name is derived from the CSV

## Part Developer User Guide

### Import and Export

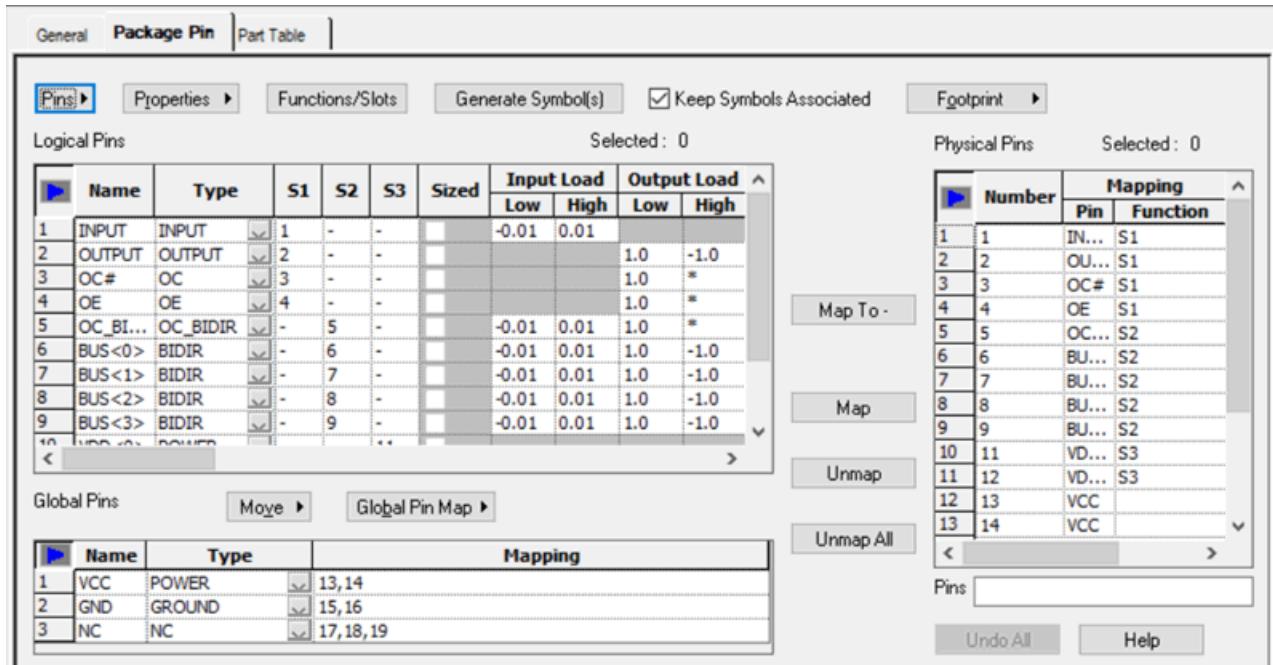
---

filename, and the package name is derived from the `package_name` property value. The `part_number` and `pack_type` properties appear as additional properties for the package.



The `jedec_type` value, `dip_20`, is assigned to the *Jedec Type* field. The `alt_symbols` value, `dip20_3`, is assigned to the *Alt Symbols* field.

The pin names and pin properties are displayed on the Package Pin page.



The CONSOLIDATED\_DIP package has three function groups, with one slot in each function group. The pins INPUT, OUTPUT, OC#, and OE are present in the first section, OC\_BIDIR and BUS<3..0> in the second section, VDD<0> and VDD<1> in the third section, and pin INVALID! is common across all sections.

## Part Developer User Guide

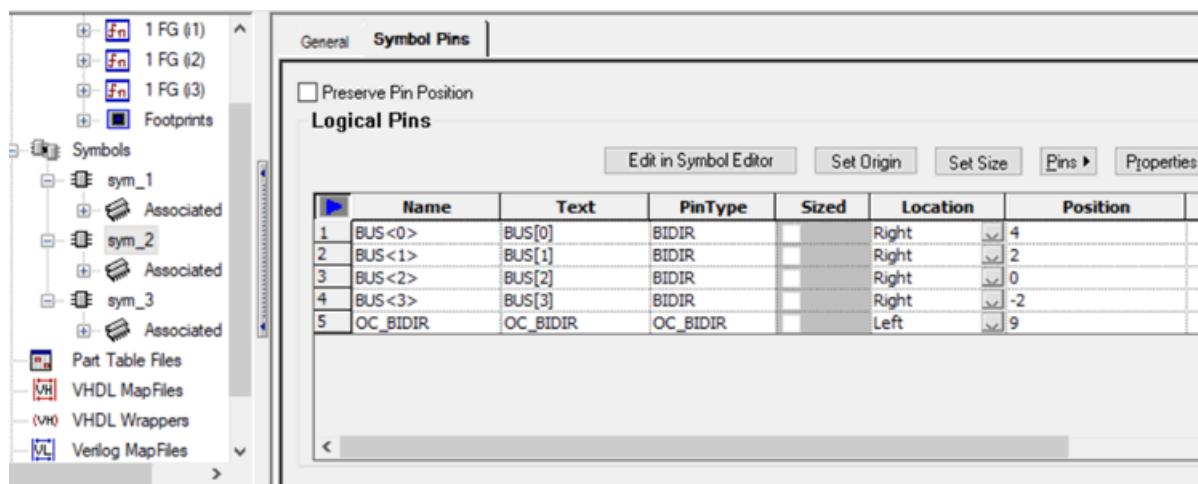
### Import and Export

The number of function groups and the number of slots in each function group are determined by the combination of the pin numbers and the symbol numbers. For example, because pin number 1 is present only in symbol 1, it is determined that the pin with name INPUT is present only in the first function group. Similarly, because the INVALID! pin does not have a symbol number associated with it, it is included in all the function groups.

The pins VCC, GND, and NC pins are included in the *Global Pins* grid because the CSV file does not have symbol information for these pins.

Three symbols are created, with *sym\_1* representing the first function group, *sym\_2* the second, and *sym\_3* the third function group.

The following graphic displays the symbol for the second function group:



The position of each symbol pin from the origin is determined from the `pin_position` values in the CSV file.

## Steps

The steps are as follows:

1. Open a project in Part Developer.
2. Choose *File – Import and Export*.  
The Import and Export *wizard* appears.
3. Choose *Import Comma Separated Value (.csv) file* and click *Next*.

The Select Source page appears.

4. Browse and select the input CSV file.

The Select Destination page appears.

The name of the part to be created is seeded automatically from the CSV file name. If required, change the part name.

5. Select the library in which the part is to be created and click *Next*.

The Preview of Import Data page appears.

To understand how Part Developer converts data from the CSV file, see [Conversion Details](#) on page 246.

6. Click *Finish* to complete the part creation process.

If there are duplicate pins in the CSV file, Duplicate Pin Resolver Dialog appears. You need to resolve the duplicate pins. For more information, see [Duplicate Pin Name Handling](#) on page 251.

Finally, the Cell Editor appears with the part information.

## Import Synopsys PTM Model

PTM Model files contain mapping information between the physical pin numbers and the LMC Swift Model ports. The PTM files are made by a tool called `ptm_make`.

### Conversion Details

The methodology used by Part Developer to create parts from Synopsys PTM Model files is as follows:

- Multiple pin to model-port sections in PTM model are created as different packages upon import.
- The package name is derived as `<model>_<package>` where model is the value of Model and package is the value of the PACKAGE property of the PTM model.
- The total number of pins is derived from the value of the PIN\_COUNT property. The pin numbers that are missing from the MODEL\_PORT entries are added as NC pins.
- The pin types are derived as follows:
  - IN as INPUT
  - OUT as OUTPUT

- IXO as BIDIR

## Example

Consider the following PTM model:

```
MODEL|TTL02
PACKAGE|DIP0
PIN_COUNT|14
DEVICE/COMP|54F02DM|54F02-FAI
DEVICE/COMP|74F02PC|74F02-FAI
MODEL_PORT|I1|IN|2|5|8|11
MODEL_PORT|I2|IXO|3|6|9|12
MODEL_PORT|O1|OUT|1|4|10|13
```

For this, a package with package name TTL02\_DIP0 will be created with logical pins I1, I2, and O1 mapped to the pin numbers in four slots. As shown above, the PIN\_COUNT property has a value of 14, which means that the package has pin numbers from 1-14. Since pin numbers 7 and 14 are not defined in the above mapping in the PTM file, they will be added as NC pins to the package.

## Steps

1. Choose *File – Import and Export*.

The Import and Export *wizard* appears.

2. Select the *Import Synopsis PTM Model* entry and click *Next*.

The Select Source page appears.

3. Browse and select the input Synopsis PTM Model file.

The Select Destination page appears.

The name of the part to be created is seeded automatically. If required, change the part name.

4. Select the library in which the part is to be created and click on *Finish*.

The Cell Editor appears with the part information.

## Import Verilog Model

Verilog models contain port information for a part. Part Developer creates symbols from Verilog model files.

### Conversion Details

- Only the first module declaration is read for the import. All other module declarations are ignored during import. A warning is written in the Session Log stating that Only the first model will be imported.
- The ports in the module declaration are used to derive the symbol pins.
- The pin types are derived from the model ports as follows:
  - INPUT as INPUT
  - OUTPUT as OUTPUT
  - INOUT as BIDIR
- The symbol is generated using the values specified in *Setup*. Pin text is assigned only if the *Use Pin Name as Pin Text* check box is selected.
- Parameters defined in the Verilog file are added as symbol properties. For example, consider a Verilog model file with the following parameter:

*parameter ModelType = "SWIFT";*

When imported, it gets added to the symbol as the following two properties:

---

Property Name	Property Value
VLOG_PARAM1	ModelType:string
ModelType	SWIFT

---

### Steps

The steps are as follows:

1. Choose *File – Import and Export*.

The Import and Export *wizard* appears.

2. Choose *Import Verilog Model* and click *Next*.

The Select Source page appears.

3. Browse and select the input Verilog model file.

The Select Destination page appears.

4. The name of the part to be created is seeded automatically. If required, change the part name.

5. Select the library in which the part is to be created and click on *Finish*.

The Cell Editor appears with the part information.

## Import VHDL Model

VHDL models contain port information for a part. Part Developer creates symbols from VHDL model files.

### Conversion Details

- Only the first entity declaration is read from the VHDL model file. A warning is written in the Session Log stating that only the first entity will be imported. All other entity declarations are ignored during import.
- The ports in the entity declaration is used to derive the symbol pins.
- The pin types are derived from the model ports as follows:
  - INPUT as INPUT
  - OUTPUT as OUTPUT
  - INOUT as BIDIR
- The symbol is generated using the values specified in *Setup*. Pin text is assigned only if the *Use Pin Name as Pin Text* option is checked.
- Library and Use clauses go into the symbol as symbol properties.
- Generics defined in the entity declaration are added as symbol properties. For example, consider the following entity declaration:

*generic {*

*tw\_ctl : time := 5 ns; -- min pulse duration for pre\* and clr\**

};

When imported, it gets added to the symbol as the following two properties:

Property Name	Property Value
VHDL_GENERIC1	tw_cntl:time
TW_CNTL	5ns

## Steps

The steps are as follows:

1. Open a project in Part Developer.
2. Choose *File – Import and Export*.  
The Import and Export *wizard* appears.
3. Choose *Import VHDL Model* and click *Next*.  
The Select Source page appears.
4. Browse and select the input VHDL model file.  
The Select Destination page appears.

The name of the part to be created is seeded automatically. If required, change the part name.

5. Select the library in which the part is to be created and click on *Finish*.

The Cell Editor appears with the part information.

## Import FPGA

Part Developer lets you use the place-and-route data created using tools from FPGA vendors Actel, Altera, and Xilinx to create a library component. You can also create a block in which the FPGA component is instantiated. You can then use the FPGA component or the block in which the FPGA component is instantiated in your design.

**Note:** Pin names with characters {, (, and [ are considered vector pin names. You can configure the `Import_FPGA_Braces_TreatedAs_Vector` directive in your local or site CPM file to remove one or more of the three characters from the default list.

## Steps

The steps are as follows:

1. Open a project in Part Developer.
2. Choose *File – Import and Export*.

The Import and Export *wizard* appears.

3. Choose *Import FPGA* and click *Next*.

The Select Source page appears.

4. Choose the vendor for the place-and-route tool you used to create the place-and-route file for an FPGA. The following vendors are supported:

- Xilinx
- AlteraQuartusII
- AlteraMaxplusII

## Part Developer User Guide

### Import and Export

---

#### Actel

##### **Do the following if you selected Actel as the vendor**

**Note:** Only .pne and .pin files can be imported.

- a. Select the Actel device family you used when creating the place-and-route file.
- b. Specify the name and path to the place-and-route file you created using the Actel place-and-route tool.
- c. Specify the name and path to the package file for the device family.

The default package file for the device family is displayed. You can specify a different package file.

- d. Specify the name and path to the pin file to be used for the Actel device family.

The default pin file for the Actel device family is displayed. You can specify a different pin file.

##### **Do the following if you selected AlteraMaxPlusII or AlteraQuardusII as the vendor**

**Note:** Only .pin files can be imported.

- Specify the name and path to the place-and-route file you created using the Altera place-and-route tool.

##### **Do the following if you selected Xilinx as the vendor**

**Note:** Only .pad and .csv files can be imported.

- Specify the name and path to the pad file you created using the Xilinx place-and-route tool.

#### 5. Click *Simulation Options* and specify the following:

- Specify the name and path to the Verilog file that contains the functional description of the FPGA. This file will be used by the simulator.
- Specify the name and path to the SDF file that contains the delay information of the FPGA.

## Part Developer User Guide

### Import and Export

---

**6.** Click Next.

The Select Destination page appears.

**7.** You can now create a new schematic symbol for the FPGA or use an existing component as the schematic symbol for the FPGA.

Select	To
Generate Custom Component	<p>Create a new schematic symbol for the FPGA.</p> <p>Click <i>Default Properties</i> to display the <i>Default Properties</i> dialog box. Specify the default values for the following properties.</p> <ul style="list-style-type: none"><li>■ DESCRIPTION</li><li>■ FAMILY</li><li>■ JEDEC_TYPE</li><li>■ PART_NUMBER</li></ul> <p>To specify the value for a property, select a property from the <i>Name</i> drop-down list, then enter its value in the <i>Value</i> field.</p>

## Part Developer User Guide

### Import and Export

---

<b>Select</b>	<b>To</b>
Use standard component	<p>Use an existing component as the schematic symbol for the FPGA.</p> <p>Select the Library in which the component exists from the <i>Library</i> drop-down list, then select the component in the <i>Cell</i> drop-down list.</p> <p><b>Note:</b> The standard component option is useful only if you are using the FPGA component in the Allegro Design Editor environment.</p>

### **Methodology**

Part Developer replicates the HDL views from the source component to the target component. In the target component, it creates the pinlist.txt file in the FPGA view. This file contains a pin name to number mapping, which is derived from the FPGA source file, such as the pad, pin, or csv file.

**Note:** The pinlist.txt file contains only the pin information for the programmable pins. Standard programming pins and power pins are not included because their names are standard and do not change when programming the FPGA.

There are some guidelines that must be followed:

- The primitive name is not changed when the source component is replicated. Therefore, it is important to ensure that the logical pin list of the destination component is always matching the source component. This is critical for the flow because if the pin list is different and the netlister finds the same primitive from two different places, then it will fail.
- The physical pin list of the target component should match the physical pin list of the FPGA view. This is required because Allegro Design Editor overlays the logical pins on the basis of the physical pin list in the FPGA view.
- In case of multiple primitives, all the primitives get copied and the pinlist.txt file is valid for all the primitives.

By default, the name of the FPGA component that will be created will be the same as the name of the place-and-route or pad file you selected in [step 4](#). You can change the name if required.

1. Select the library in which you want the FPGA component to be created.

2. Click *Next*.

The Preview of Import Data page appears.

3. Click *Finish*.

The Errors & Warnings page appears displaying the status of the FPGA import.

You can now use Part Information Manager to add the FPGA component or the block instantiating the FPGA component in your design.

## Location of Actel FPGA Libraries

The following table lists the location of Actel FPGA libraries. These libraries are available with your Cadence software installation.

---

Type of Files	Location <sup>1</sup>
Location of package files for Actel device families	\$CADENCE/share/library/actel/data/ <device_family>.pkg
Location of the default PGA pin file for Actel device families	\$CADENCE/share/library/actel/data/actel.pga.pin

---

1. Where \$CADENCE is the Cadence installation directory.

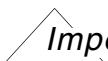
## Import Text File

The *Import Text File* option enables you to import part information from a delimiter-separated text file to create a part.

### Conversion Details

- The following headers are read and translated during text import:
  - pin\_name

- pin\_number
- pin\_type
- pin\_location
- pin\_position
- symbol

 *Important*

If required, these header keywords can be changed to your specifications. See the **Configuring the Pre-Defined Headers for Text Import** section in the **Advanced Tasks** chapter in the **Part Developer User Guide**.

- The minimum headers required are as follows:
  - pin\_name for flat or single-section parts
  - pin\_name, pin\_number, and symbol for multi-section parts

The package and symbol information is determined in the following way:

- By default, the package name is the same as the cell name to be created. It can be changed on the *Select Destination* page during text import.
- Entries under the pin\_name column are used as pin names.
- Entries under the pin\_number column are used as pin numbers.
- Entries under the pin\_type column are used as pin types. Various pin types are automatically converted to supported pin types by using the definitions provided in the propfile.prop file. For example, if the pin type IO is specified in the text file, it is converted to BIDIR.
- Entries under the pin\_location column are used to determine the location of pins (left, right, top, bottom, left-right, and any) on the symbol. The left-right and any options help to generate symbols of large pin-count parts in such a way that, if possible, they can be displayed on the same sheet. The following table explains the left-right and any options:

---

left-right	Pins are placed on left and right sides of the symbol in a round-robin fashion
any	Pins are placed on left, right, top, and bottom sides of the symbol in a round-robin fashion

---

## Part Developer User Guide

### Import and Export

---

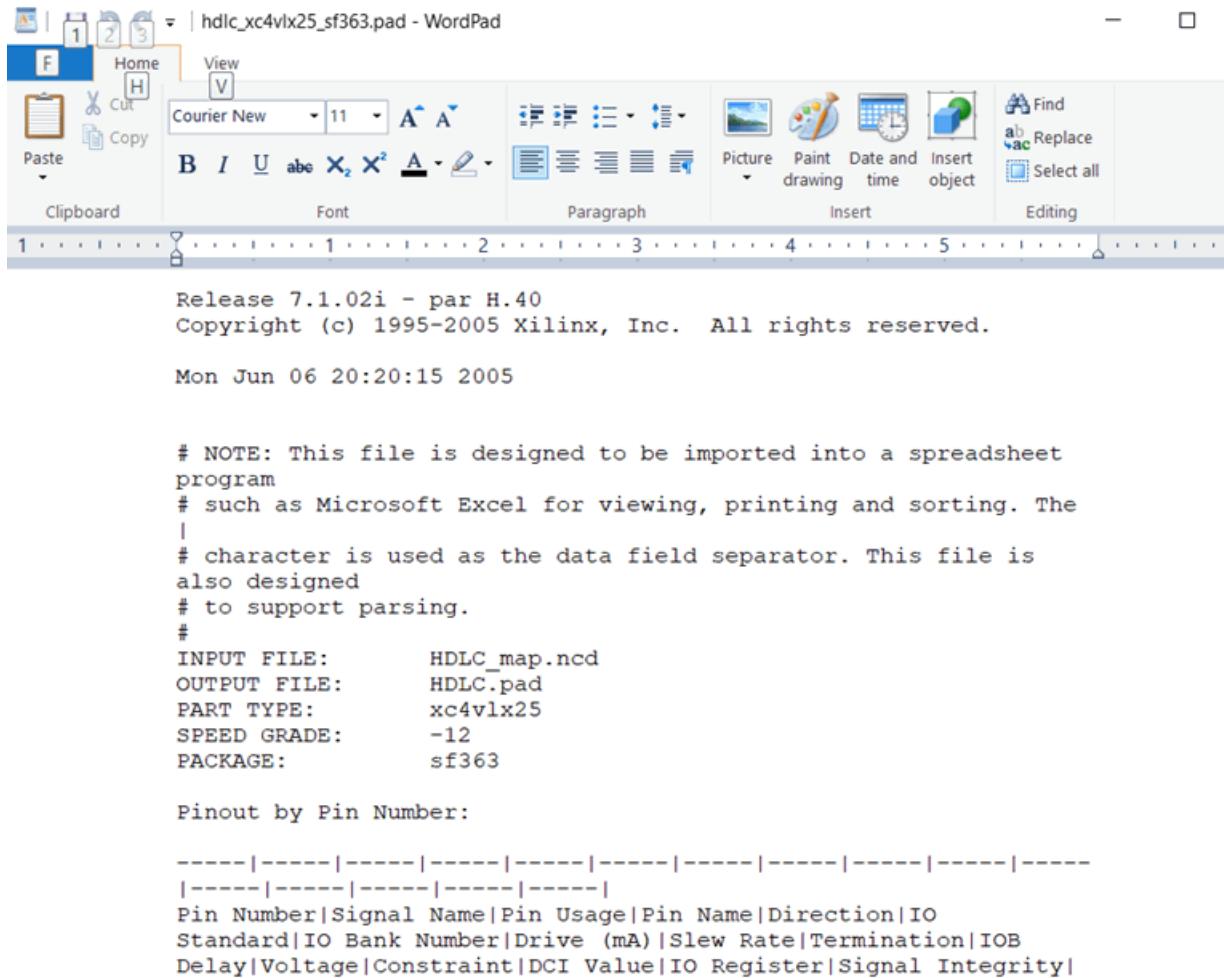
- Entries under the pin\_position column are used to determine the position of pins with respect to the origin of the symbol. A pin\_position column value appears as the value of the *Position* column in the Symbol Pins panel.
- If the symbol is not present as a column in the text file, the symbol is created only when the *Generate Symbol* option is selected on the Select Views page.
- If the symbol column value is not specified for a POWER, GROUND, or NC pin, the pin is moved to the global section. Otherwise, it is moved to the logical section.
- If duplicate pins exist in the text file, the Duplicate Pin Resolver is used as in CSV import to resolve the problem. For more details, see [Duplicate Pin Name Handling](#) on page 251.
- If a pin exists on multiple symbols with different pin types, one of the pin types is used and the rest is ignored. You should correct the text data before the import process.
- Pin names with invalid characters in them are automatically converted to valid pin names according to CSV rules by using the entries provided in the `translate.cpm` file, which is located at `<your_inst_dir>\share\cdssetup\lman`. For example, the character ! is translated to \_EXCL\_ because it is not a supported character in the Cadence flow.
- If the `Import_Text_Pins_DefaultVector` directive is set to TRUE, pin names with ( ) or { } or [] are treated as vector else they are treated as scalar.
- The directive `Import_Text_Braces_TreatedAs_Vector '{,(),['` is used to specify which braces to be used for vector notation and is dependent on the `Import_Text_Pins_DefaultVector` directive being set to TRUE. By default, {}, (), and [] are treated for vector notation. You can configure the `Import_Text_Braces_TreatedAs_Vector` directive in your local or site CPM file to remove one or more of the three characters from the default list.

## **Part Developer User Guide**

### Import and Export

## Example

Consider the file `HDLC_xc4v1x25_sf363.pad`, which has the following entries:

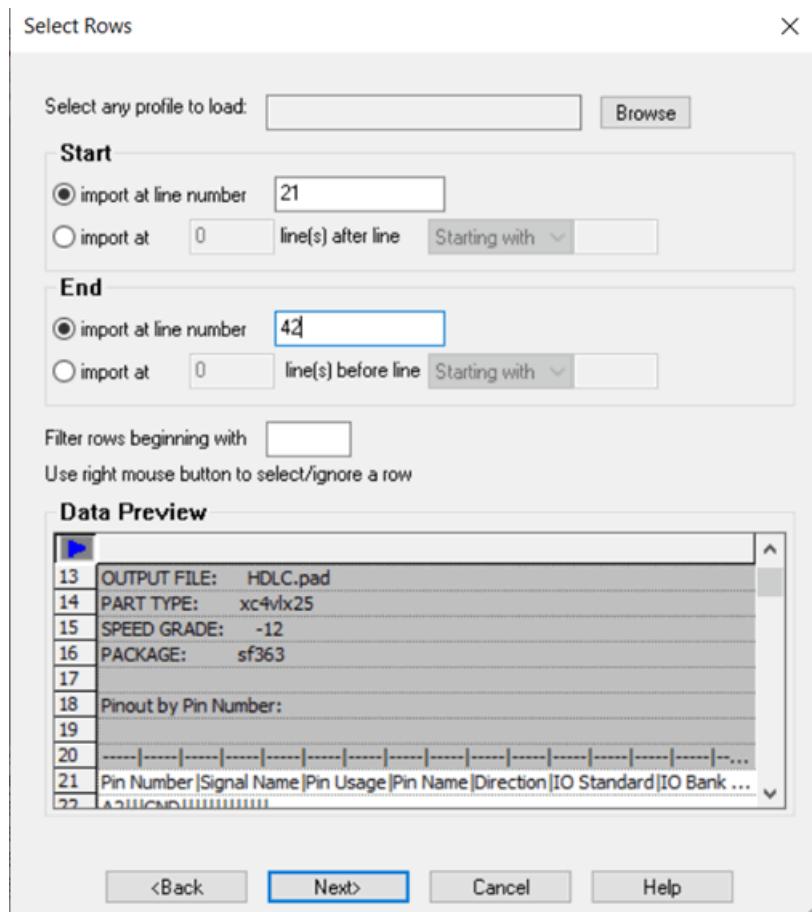


When you import this text file in Part Developer, a part with the same name as the text file is created unless you specify a different name during the import process. Notice that the pin

## Part Developer User Guide

### Import and Export

information in the text file is preceded by 20 lines of text, which needs to be ignored in the import process. To ensure this, specify the start import line number as 21.

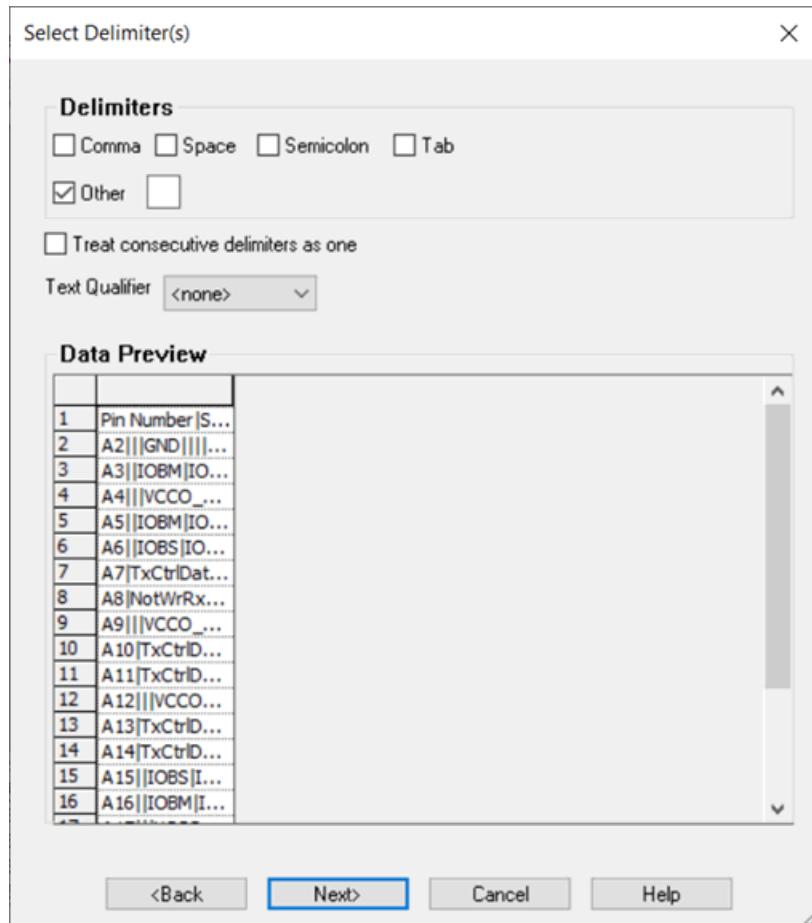


If you set the first row of the text file as the header, the text import wizard recognizes PIN\_NUMBER, PIN\_NAME, PIN\_TYPE, PIN\_LOCATION, and PIN\_POSITION columns and translates them to appropriate properties. However, the last two are used only if you generate the symbol for the part using the text import wizard.

## Part Developer User Guide

### Import and Export

The delimiter used in the HDLC\_xc4vlx25\_sf363.pad file is |. When you specify the delimiter, the text import wizard parses the data and displays the data in a tabular format for preview.



Notice the *Treat consecutive delimiters as one* option. If your text file has consecutive delimiters, selecting this option treats multiple occurrences of the delimiter as one occurrence.

The *Text Qualifier* option enables you to import values that contain the characters you have specified as delimiters. Take the example of INPUT\_LOAD and OUTPUT\_LOAD values, such as "(1.0,1.0)" and "(0.01;0.01)". To import these values, you need to specify comma and semicolon, respectively, as a text qualifier. Currently, Part Developer supports only comma and double quotation mark as text qualifiers.

## Profile Use Model

Profiles enable you to save your preferred settings for filtering the import data so that the same settings can be applied in later sessions of import by simply loading the profile files.

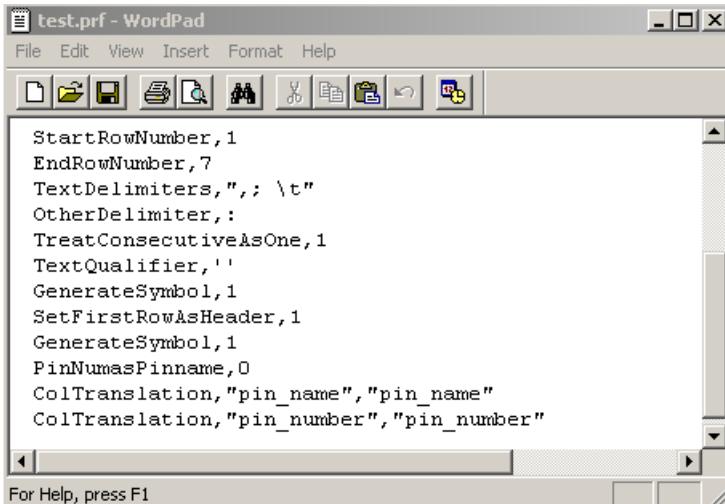
## Part Developer User Guide

### Import and Export

The following table lists the settings that are saved in a profile file and the corresponding directives:

Settings	Directives
Range of data to be imported as specified using the options in Start and End areas	<b>StartRowNumber</b> <b>EndRowNumber</b>
Delimiters to be used to parse data	<b>TextDelimiters</b> <b>OtherDelimiter</b>
Whether or not consecutive delimiters are to be treated as one	<b>TreatConsecutiveAsOne</b>
Text qualifier if any	<b>TextQualifier</b>
Whether or not a symbol is to be generated	<b>GenerateSymbol</b>
Whether or not the first row is to be set as the header	<b>SetFirstRowAsHeader</b>
Whether or not pin numbers are to be set as pin names	<b>PinNumasPinname</b>

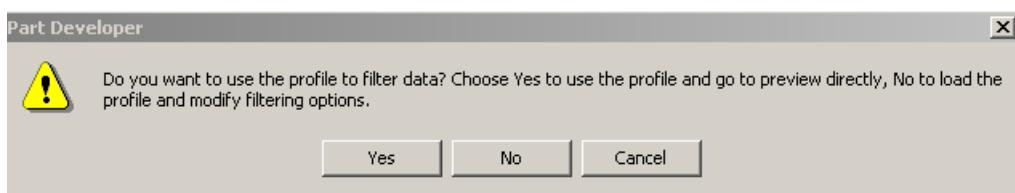
The following example shows how various directives are specified in a profile file:



```
test.prf - WordPad
File Edit View Insert Format Help
For Help, press F1
StartRowNumber,1
EndRowNumber,7
TextDelimiters,"; \t"
OtherDelimiter,: 
TreatConsecutiveAsOne,1
TextQualifier,''
GenerateSymbol,1
SetFirstRowAsHeader,1
GenerateSymbol,1
PinNumasPinname,0
ColTranslation,"pin_name","pin_name"
ColTranslation,"pin_number","pin_number"
```

 *Important*

The selection of rows and columns made in the Data Preview area by using the pop-up menu and the footprint specified for the part are not saved in the profile file. Therefore, if you want to add a footprint, you should choose the *No* option to load the profile file and modify the filtering options when Part Developer displays the following message:



In this way, the profile settings will be loaded and the text import wizard will not jump to the Preview of Derived Data page. You can then step through all the text import wizard pages and modify any profile settings if necessary.

## Steps

The steps to import a text file are as follows:

1. Open a project in Part Developer.
2. Choose *File – Import and Export*.

The Import and Export *wizard* page appears.

3. Click the *Import Text File* option and click *Next*.

The Select Source page appears.

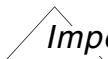
4. Browse to select the text file you want to import and click *Next*.

The Select Destination page appears.

The name of the part to be created is seeded automatically from the text file name. If required, change the name.

5. Select the library in which the part is to be created and click *Next*.
6. On the Select Rows page, specify the rows in the input file that should be imported and click *Next*.

You can preview the contents of the text file to be imported in the Data Preview area of the Select Rows page. To know more about the options on the Select Row page, see [Select Rows on page 432](#).

 **Important**

If you want to seed in your format-specific preferences, you can do so by specifying the name and the location of the profile file in the *Select any profile to load* field. If you select a profile, you can directly jump to [step 16](#). To know more about profiles, see [Profile Use Model on page 269](#).

7. On the Select Delimiter(s) page, choose the delimiter that should be used to parse the import data and if required, select the *Treat consecutive delimiters as one* check box. Based on your selection, the data is parsed and displayed in rows and columns in the *Data Preview* area.
8. From the *Text Qualifier* drop-down list, select any character that you do not want Part Developer to treat as a delimiter and click *Next*.

The Select Columns page appears. To know more about the options on the Select Columns page, see [Select Columns on page 434](#).

9. Select the *Set first row as header* check box if required.
10. Select the *Set pin number as pin name* check box.

Since the pin name column is the only column required for the import process, selecting this option ensures that the import process is successful if the text file contains only the pin number column.

11. In the Data Preview area, choose the columns for import and click *Next*.

The Select Views page appears. To know more about the options on the Select Views page, see [Select Views on page 435](#).

12. Select the *Generate Symbol* check box.
13. Select the *Add Footprint* check box and choose a footprint from the list of footprints displayed from PSMPATH.
14. Click the *Save Profile As* button and specify the name of the file in which you want Part Developer to save your format-specific preferences.

**Note:** Your preferences are saved in a `.prf` file.

15. To move to the final step in the import process, click *Next*.

The Preview of Derived Data page appears. You can preview the data in *Logical Pins* and *Global Pins* grids.

- 16.** Click *Finish* to complete the import process.

The Cell Editor appears with the part information.

## Import ViewLogic(VL) Part

The *Import ViewLogic Files* option enables the creation of parts from a ViewLogic file.

### Conversion Details

ViewLogic provides part information through files named as `<partname>.n`. For example, `mycell.1`, `mycell.2`, `mycell.3`, and `mycell.4` where each one is a particular version of the `mycell` part. The part information itself is specified through attributes. Part Developer reads all parts and their corresponding attributes to create the chips and the symbols for the part.

The methodology used by Part Developer to create parts from ViewLogic files is as follows:

- The values of the `PARTS` and `#` attributes are used to determine the number of sections for the part. For example, `PART=4` and `#=1,4,9,12` will be interpreted as a part with 4 slots with the pin being mapped to physical pins 1, 4, 9, and 12 in the four slots.
- The value of the `ALT_SYMBOL` attribute is used to determine the `ALT_SYMBOLS` property. ViewLogic follows the standard Allegro syntax for defining the value of this property. Therefore, the value is taken as is.
- The value of the `CLASS` attribute is used to determine the `CLASS` property. ViewLogic supports a large number of values for the `CLASS` attribute. However, Design Entry HDL supports only four different values for the `CLASS` property: `DISCRETE`, `IO`, `IC`, and `MECHANICAL`. Therefore, Part Developer automatically maps the `CLASS` attribute values to the `CLASS` property values supported in Design Entry HDL. The mapping is done as per the following table:

<b>ViewLogic</b>	<b>Design Entry HDL</b>
ADC	IC
ANALOG	IC
C	DISCRETE

## Part Developer User Guide

### Import and Export

---

CMOS	IC
CRYSTAL	DISCRETE
DAC	IC
DIODE	DISCRETE
ECL	IC
FET	DISCRETE
FILT	DISCRETE
L	DISCRETE
MOS	IC
NPN	DISCRETE
P-C	DISCRETE
PNP	DISCRETE
POT	DISCRETE
R	DISCRETE
RP	DISCRETE
SCR	DISCRETE
SDIODE	DISCRETE
SWITCH	DISCRETE
TRANSFOR	DISCRETE
VR	DISCRETE
ZENER	DISCRETE
TRANSFOR	DISCRETE
VR	DISCRETE
ZENER	DISCRETE

- The value of the DEVICE attribute is used to derive the primitive name. If the value is ?, then the file name will be used as the primitive name.
- The presence of the HETERO attribute results in the creation of a split part. By default, ViewLogic supports four different syntax for the HETERO attribute to specify different types of splits.

## Part Developer User Guide

### Import and Export

---

Hetero Type 1 - Different Components Within the Same,  
HETERO=symname1, symname2 [, symname3....]

Hetero Type 2 - Different Gates Within the Same Device,  
HETERO=symname, (symnameP)

Hetero Type 3 - Split ICs,  
HETERO=(icsymname1),(icsymname2)[,(icsymname3)..]

Hetero Type 4 - Different Representations of the Same Device,  
HETERO=(symname1), symname

Part Developer creates parts as following:

- In Hetero Type 1, each symname represents one or more sections of a single package.
- In Hetero Type 2, each symname causes the creation of a package.
- In Hetero Type 3, each symname represents one section of a split part.
- In Hetero Type 4, each symname causes the creation of a package.
- The LEVEL attribute value determines whether the part will be read by Part Developer. Part Developer imports data only if the attribute has the value STD.
- The value of the NC attribute is used to derive the NC pins for the part.
- The PART\_SPEC attribute is taken as is on the symbol as a property. Alternatively, this attribute can also be moved into the chips.prt file by writing appropriate translation rules.
- The P/D\_NUM attribute is taken as is on the symbol as a property.
- The PINORDER attribute is taken as is on the symbol as a property.
- The PINSWAP attribute supports parentheses and brackets in its value string. Parentheses is used to indicate swappable groups and brackets to indicate groups that are not swappable. Currently, only the parentheses are translated, and Part Developer uses them to determine the value of the PIN\_GROUP property in the chips.prt file.
- The value of the PINTYPE attribute is used to determine the pin type. The translation happens as per the following table:

---

**ViewLogic**

**Design Entry HDL**

---

ANALOG

ANALOG

## Part Developer User Guide

### Import and Export

---

<b>ViewLogic</b>	<b>Design Entry HDL</b>
BI	BIDIR
IN	INPUT
OCL	OC
OEM	OE
OUT	OUTPUT
TRI	TS

---

The property translation is done as per the following table:

<b>ViewLogic Property</b>	<b>Part Developer Property</b>
PKG_TYPE	JEDEC_TYPE
REFDES	PHYS_DES_PREFIX
SIGNAL	POWER_PINS
TOLERANCE	Moved as is to the chips.prt file.
VALUE	Moved as is to the chips.prt file.
VDD_PIN	VDD power pins

---

## Steps

The steps are as follows:

1. Choose *File – Import and Export*.  
The Import and Export wizard appears.
2. Choose *Import ViewLogic(VL) Part* and click *Next*.  
The Select Source page appears.
3. Browse and select the directory in which the ViewLogic files exist and click *Next*.  
The Select Package page appears.
4. Select the part that you want to import and click *Next*.

The Select Destination page appears.

5. Enter the name of the cell to be created and the library in which it should be created and click *Finish*.

The Cell Editor appears with the part information.

## Import Allegro Footprint

The *Import Allegro Footprint* option enables the creation of parts from Allegro footprints.

### Conversion Details

- The pin names are derived from the physical pin numbers of the imported footprint.
- The default pin type is BIDIR. However, it can be configured using the cpm directive `Import_AllegroFtprint_DefaultPinType 'BIDIR'`.
- By default, the cell name and the part name are the same as the name of the footprint being imported.

### Steps

The steps are as follows:

1. Choose *File – Import and Export*.

The Import and Export wizard appears.

2. Choose *Import Allegro Footprint* and click *Next*.

The Select Footprint page displays all of the footprints in `PSMPATH`.

3. Select the footprint you want to import and click *Next*.

The Select Destination page appears.

4. Enter the name of the cell to be created and the library in which it should be created and click *Next*.

The Preview of Import Data page appears. You can preview the data in *Logical Pins* and *Global Pins* grids.

5. Click *Finish* to complete the import process.

The Cell Editor appears with the part information.

## Import Die Text

The *Import Die Text* option enables you to import part information from a die file to create a part. The die file format is the standard format generated by APD.

### Conversion Details

- The following headers are read from the die file:
  - Pin Number
  - Pin Use
  - Pin Name
  - Swap Code
- The minimum headers required are as follows:
  - Pin Number and Pin Name
- The Pin Number column must be the first column.
- As Part Developer supports the entire APD-supported character set, no conversion is required for valid characters. However, if the die file being imported contains any invalid character in pin names, the pin names are translated to valid names.
- The character \* is treated as the low-assertion character.

### Steps

The steps are as follows:

1. Choose *File – Import and Export*.  
The Import and Export wizard appears.
2. Choose *Import Die Text* and click *Next*.  
The Select Source page appears.
3. Browse to locate the die file and click *Next*.  
The Select Destination page appears.

The name of the part to be created is seeded automatically from the name of the die file. If required, change the part name.

4. Enter the cell name, select the destination library, and click *Next*.

The Preview of Derived Data page appears.

5. Click *Finish* to complete the part creation process.

If there are duplicate pins in the die file, Duplicate Pin Resolver Dialog appears. You need to resolve the duplicate pins. For more information, see [Duplicate Pin Name Handling](#) on page 251.

Finally, the Cell Editor appears with the part information.

 **Important**

*Import Die Text* uses a predefined profile called `importApdDieText.prf`, which you can modify according to your requirements. The `importApdDieText.prf` file is located at `<your_inst_dir>\share\cdssetup\LMAN`.

 **Important**

To import die files from the command prompt, you can use the `text2con` command with the `importApdDieText.prf` profile file. For the syntax of the `text2con` command, see [text2con](#) on page 404.

## Import DML Model

The *Import DML Model* option enables the creation of parts from a DML model.

### Conversion Details

- Translation rules are applied as per the directives set in the `translate.cpm` file located at `<install_dir>/share/cdssetup/LMAN`.
- If a DML file has models that have no signal model associated to a signal name other than NC, then these models are created as NC pins.
- If the signal name has non-numeric characters inside ( ), {}, or [], then the names are truncated on IBIS / DML / SI import. For example, signal names such as abc(e13f) or abc{e13f} or abc[e13f] will get imported as abc.
- If the `Import_DML_Pins_DefaultVector` is set to TRUE, then the signal name having ( ) or { } or [] will be treated as vector else it will be treated as scalar.
- The directive `Import_DML_Braces_TreatedAs_Vector ' { , ( , [ '` is used to specify which braces to be used for vector notation and is dependent on the

Import\_DML\_Pins\_DefaultVector directive being set to TRUE. By default, '{, (, [ are treated for vector notation.

## Steps

The steps are as follows:

1. Choose *File – Import and Export*.

The Import and Export wizard appears.

2. Choose *Import DML Model* and click *Next*.

The Select Source page appears.

3. Browse and select the directory in which the DML model files exist and click *Next*.

**Note:** Select *Group all primitives in one cell* to include all the package information into the chips.prt file.

The Select Device page appears.

4. Select the device type to be imported and click *Next*. By default, this will display only the model name if *Group all primitives in one cell* was selected on the previous page.

5. Enter the name of the cell to be created and the library in which it should be created and click *Finish*.

The Cell Editor appears with the part information.

## Import IBIS Model

The *Import IBIS Model* option enables the creation of parts from an IBIS model.

**Note:** Part Developer internally converts the IBIS model to a DML model before converting it to a part.

**Note:** Some of the programs required for this functionality are not installed. Install product PX3120 Allegro PCB Model Integrity for the missing programs. A license for PX3120 is not required for executing the programs needed for this functionality.

## Conversion Details

- Translation rules are applied as per the directives set in the `translate.cpm` file located at `<install_dir>/share/cdssetup/LMAN`.

- If an IBIS model has a pin with model name as NC but a signal name other than NC, then these models are created as NC pins.
- If the signal name has non-numeric characters inside the ( ) or {} or [], the names are truncated on IBIS / DML / SI import. For example, signal names such as abc(e13f) or abc{e13f} or abc[e13f] will get imported as abc.
- The value of the `Import_IBIS_With_Ibischk4` directive determines whether `ibischk4` will be run during conversion. The possible values are TRUE and FALSE. By default, this directive is set to TRUE.
- The value of the `Import_IBIS_With_Dmlcheck` directive determines whether DML checks will be run on the converted DML file. The possible values are TRUE and FALSE. By default, this directive is set to TRUE.
- The value of the `Import_IBIS_With_Uncanged_ModelName` directive determines how the model name is written in the DML file. If this is set to FALSE, the model names in the DML file will be written as `<Ibis_file_name>_<Model_name_in_IBIS_FILE>`.

## Steps

The steps are as follows:

1. Choose *File – Import and Export*.

The Import and Export wizard appears.

2. Choose *Import IBIS Model* and click *Next*.

The Select Source page appears.

3. Browse and select the directory in which the IBIS model files exist and click *Next*.

**Note:** Select *Group all primitives in one cell* to include all the package information into the chips.prt file.

The Select Package page appears.

4. Select the device type to be imported and click *Next*. By default, this will display only the model name if *Group all primitives in one cell* was selected in the previous page.
5. Enter the name of the cell to be created and the library in which it should be created and click *Finish*.

The Cell Editor appears with the part information.

## Import Mentor Part

The *Import Mentor Part* option enables the creation of Design Entry HDL parts from Mentor parts.

**Note:** A valid Mentor DA license and installation must be available in the path for successful import of Mentor symbols. Both Cadence and Mentor call their DA tool `da.exe`. Therefore, when converting from Mentor to Design Entry HDL parts, it is necessary to have the Mentor DA tool in path first.

### Conversion Details

#### Parts Translation

All types of Mentor parts (homogenous and heterogenous) can be translated. The translation is done as per the following rules:

- The logical symbol and map file corresponding to a Mentor part translate to `symbol.css` and `chips.prt`, respectively.
- All the different symbols or views of the symbol corresponding to a Mentor part are translated in one go along with the specified map file.

#### Property Translation

Property translation is done as per the following table:

Mentor Part Property	Part Developer Property
DC_WORK_VOLTAGE	VOLTAGE
GEOM	JEDEC_TYPE
INSTPAR	VALUE
NET	SIG_NAME
NET_TYPE	NET_PHYSICAL_TYPE
PIN_NO	\$PN
POWER_NETS	POWER_GROUP

## Part Developer User Guide

### Import and Export

---

REF	(\$LOCATION
TOLER	TOL

All other Mentor properties are created as is in Part Developer.

**Note:** You can override the standard translation processing by providing a property map, which dictates how properties should be translated.

### Pin Translation

Pin translation is done as per the following table:

Mentor Pin Type	Design Entry HDL Pin Type
IN	INPUT
OUT	OUTPUT
IXO	BIDIR
AN	BIDIR

For example, the bus pin name is changed according to the Design Entry HDL bus pin naming convention, such as "MY\_NET(2 : 0)" (or "MY\_NET[2 : 0]") would be translated to "MY\_NET<2 .. 0>".

### Shape Support

Except filled shapes, Part Developer supports all Mentor-supported shapes.

The shape translation is done as per the following table.

Mentor Shape	Design Entry HDL Shape
Arc	Arc
Circle	Arc
Line	Line
Polygon	Lines

## Part Developer User Guide

### Import and Export

---

Polyline	Lines
Rectangle	Lines
Text	Text
Filled circle	Arc

### Color Support

Mentor supports 74 different types or shades of colors. Part Developer supports 73 different types or shades of colors. Mentor colors are mapped to the nearest Part Developer-supported color by the values of RGB. The mapping can be overridden by the user as this is configurable.

### Catalog Support

Part Developer supports all the Mentor catalog formats except the now obsolete LMS format.



The value of the MENTOR\_VERSION directive in the `translate.cpm` file (located at `<CDS_SITE>/share/cdssetup/LMAN`) contains the Mentor version information. By default, Mentor version 8.10 is supported. To support other versions, you need to modify the value of the MENTOR\_VERSION directive in the `translate.cpm` file. For example, if the 8.9 version is to be supported, the MENTOR\_VERSION directive should be:

```
MENTOR_VERSION '8_9'
```

Cadence strongly recommends you to use the Mentor to Design Entry HDL conversion on Windows platform with Mentor 8.10 release.

### Steps

The steps are as follows:

1. Choose *File – Import and Export*.  
The Import and Export wizard appears.
2. Choose *Import Mentor Part* and click *Next*.  
The Select Source page appears.

3. Select the component from the catalog or select the Mentor component directly from the directory and click *Next*.

The Select Destination page appears.

4. Enter the name of the cell to be created and the library in which it should be created and click *Finish*.

The Cell Editor appears with the part information.

**Note:** Mentor products are available only on Windows, Solaris, and HP-UX. Therefore, Part Developer provides the Import Mentor feature only on the above-mentioned platforms.

## Import Pin Grid

See [Importing Pin Grid](#) on page 150 for more details.

## Import Pin Table

See [Importing Pin Table](#) for more details.

## Import ECO - APD Component Files

The Import APD Component files option imports files generated by the allegro\_component command in APD to modify a Design Entry HDL part.

### Conversion Details

The allegro\_component command creates a component folder, which contains the physical, logical, and pin delay information for a part. Part Developer reads these files to modify the part.

### Steps

The steps are as follows:

1. Choose *File – Import and Export*.

The Import and Export dialog box appears.

2. Choose *Import ECO - APD Component Files* and click *Next*.

The Select Source page appears.

3. Browse to the *component* folder.
4. Select the *ECO only the pin delay values* option. This option ECOs only the pin delay values from the package\_pin\_delay file. This is done on the basis of physical pin numbers only and will ignore the logical pin names.

5. Click *Next*.

The Select Destination page appears.

6. Select the destination library and the cell to be modified and click *Next*.

The Cell Data page appears.

7. If required, you can make changes before the part is modified.

The *ECO Messages* page appears. You can view the complete list of ECO modifications that are going to be done on the input cell. By default, Part Developer deletes the properties that are present in the cell but not in the input file and retains any symbol graphic modifications that have been done during ECO. You can turn off these default behaviors by using the *Property deletions* option and the *Graphic modifications* option in the Ignore section.

8. Click *Finish* to complete the part modification.

## Import ECO - Capture Part (Windows Only)

The Import ECO - Capture option enables you to use Capture part data to modify an existing part.

### Steps

The steps are as follows:

1. Choose *File – Import and Export*.

The Import and Export dialog box appears.

2. Choose *Import ECO Capture Part (Windows Only)* and click *Next*.

The Select Source page appears.

3. Specify the Capture library and click *Next*.

The Select Capture Part page appears.

4. Select the Capture part from the *Part* drop-down list box.

**Note:** If the selected part has aliases, they get displayed in the *Aliases* list box. By default, only the master components are shown in the *Part* drop-down list box. Therefore, in case you do not find a part listed in the drop-down list box, you need to find its master component and then do the ECO.

5. Select the Design Entry HDL library and the part on which ECO is to be done and click *Next*.

The *ECO Messages* page appears. You can view the complete list of ECO modifications that are going to be done on the input cell. By default, Part Developer deletes the properties that are present in the cell but not in the input file and retains any symbol graphic modifications that have been done during ECO. You can turn off these default behaviors by using the *Property deletions* option and the *Graphic modifications* option in the Ignore section.

6. Click *Finish*.

## Import ECO - EDAXML Part

The Import ECO - EDAXML Part option enables you to use part data in EDAXML format to modify an existing part.

**Note:** The Setup values get applied to the imported part.

### Steps

The steps to create a part from an XML datasheet are:

1. Choose *File – Import and Export*.

The Import and Export dialog box appears.

2. Choose *Import ECO - EDAXML Part* and click *Next*.

The Select Source page appears.

3. Specify the XML file and click *Next*.

The Select Destination page appears.

4. Select the Design Entry HDL library and the component on which to do the ECO and click *Next*.

The *ECO Messages* page appears. You can view the complete list of ECO modifications that are going to be done on the input cell. By default, Part Developer deletes the properties that are present in the cell but not in the input file and retains any symbol graphic modifications that have been done during ECO. You can turn off these default behaviors by using the *Property deletions* option and the *Graphic modifications* option in the Ignore section.

5. Click *Finish*.

## Import ECO - Si2 PinPak XML Part

The Import ECO - Si2 PinPak XML option enables you to use the Si2PinPak XML data to modify an existing part.

### Steps

1. Open a project in Part Developer.
2. Choose *File – Import and Export*.  
The Import and Export *wizard* appears.
3. Choose *Import ECO - Si2 PinPak XML Part* and click *Next*.  
The Select Source page appears.
4. Browse and select the input Si2 PinPak file.  
The Select Destination page appears.
5. Select the destination library and the part that is to be modified and click on *Next*.

The *ECO Messages* page appears. You can view the complete list of ECO modifications that are going to be done on the input cell. By default, Part Developer deletes the properties that are present in the cell but not in the input file and retains any symbol graphic modifications that have been done during ECO. You can turn off these default behaviors by using the *Property deletions* option and the *Graphic modifications* option in the Ignore section.

6. Click *Finish*.

The Cell Editor appears with the part information.

## Import ECO - Comma Separated Value (.csv) File

The *Import ECO - Comma Separated Value (.csv)* file option enables you to modify existing part data using part information available in csv format. Shape information can also be ECOed.

### Steps

1. Open a project in Part Developer.
2. Choose *File – Import and Export*.  
The Import and Export *wizard* appears.
3. Choose *Import ECO - Comma Separated Value (.csv)* file and click *Next*.  
The *Select Source* page appears.
4. Browse and select the input CSV file.  
The Select Destination page appears.
5. Select the library and part to be modified and click *Next*.

If there are duplicate pins in the CSV file, Duplicate Pin Resolver Dialog appears. You need to resolve the duplicate pins. For more information, see [Duplicate Pin Name Handling](#) on page 251.

6. Click *Next*.

The *ECO Messages* page appears. You can view the complete list of ECO modifications that are going to be done on the input cell. By default, Part Developer deletes the properties that are present in the cell but not in the input file and retains any symbol graphic modifications that have been done during ECO. You can turn off these default behaviors by using the *Property deletions* option and the *Graphic modifications* option in the Ignore section.

7. Click *Finish*.

The Cell Editor appears with the part information.

## Import ECO - Synopsis PTM

The Import ECO-Synopsis PTM option enables you to modify part data by using part data information available through Synopsis PTM model files.

## Steps

1. Choose *File – Import and Export*.

The Import and Export *wizard* appears.

2. Choose *Import ECO - Synopsis PTM Model* and click *Next*.

The Select Source page appears.

3. Browse and select the input Synopsis PTM Model file.

The Select Destination page appears.

4. Select the library and part to be modified and click *Next*.

The *ECO Messages* page appears. You can view the complete list of ECO modifications that are going to be done on the input cell. By default, Part Developer deletes the properties that are present in the cell but not in the input file and retains any symbol graphic modifications that have been done during ECO. You can turn off these default behaviors by using the *Property deletions* option and the *Graphic modifications* option in the Ignore section.

5. Click *Finish*.

The Cell Editor appears with the part information.

## Import ECO - FPGA

The Import ECO - FPGA option enables you to modify part data by using part data information available through FPGA components.

**Note:** Please read the methodology for the standard component in the Import FPGA section before doing ECO.

When doing ECO on standard components, the FPGA view is ECOed on the basis of the FPGA file whereas all other views are ECOed as per the source component views.

## Steps

1. Choose *File – Import and Export*.

The Import and Export *wizard* appears.

2. Choose *Import ECO - FPGA* and click *Next*.

The Select Source page appears.

3. Select the input FPGA model.

The Select Destination page appears.

4. Select the library and part to be modified and click *Next*.

The *ECO Messages* page appears. You can view the complete list of ECO modifications that are going to be done on the input cell. By default, Part Developer deletes the properties that are present in the cell but not in the input file and retains any symbol graphic modifications that have been done during ECO. You can turn off these default behaviors by using the *Property deletions* option and the *Graphic modifications* option in the Ignore section.

5. Click *Finish*.

The Cell Editor appears with the part information.

## Import ECO - Text File

The *Import ECO - Text File* option enables you to modify existing part data using part information available in text files.

### Steps

1. Choose *File – Import and Export*.

The Import and Export *wizard* appears.

2. Choose *Import ECO - Text File* and click *Next*.

The Select Source page appears.

3. Browse and select the input text file.

The Select Destination for ECO page appears.

4. Select the library and part to be modified and click *Next*.

The Select Rows page appears.

Notice that the text file data is displayed in the *Data Preview* area.

5. Select a profile if you want Part Developer to seed in your format-specific preferences, specify the rows to be imported, and click *Next*.

The Select Delimiter(s) page appears.

6. Choose the delimiter that should be used to parse the import data and if required, select the *Treat consecutive delimiters as one* check box. Based on your selection, the data is parsed and displayed in rows and columns in the *Data Preview* area.
7. From the *Text Qualifier* drop-down list, select any character that you do not want Part Developer to treat as a delimiter and click *Next*.

The Select Columns page appears.

8. Select the *Set first row as header* check box if required.
9. Select the *Set pin number as pin name* check box.
10. In the *Data Preview* area, choose the columns for import and click *Next*.

The Select Views page appears.

11. Select the *Generate Symbol* check box.
12. Select the *Add Footprint* check box and specify a footprint.
13. Click the *Save Profile As* button, specify the name of the file in which you want Part Developer to save your format-specific preferences, and click *Next*.

The Preview of Derived Data page appears.

You can preview the data in *Logical Pins* and *Global Pins* grids.

14. Click *Next* to continue.

The ECO Messages page appears.

You can view the complete list of ECO modifications that are going to be done on the input cell. By default, Part Developer deletes the properties that are present in the cell but not in the input file and retains any symbol graphic modifications that have been done during ECO. You can turn off these default behaviors by using the *Property deletions* option and the *Graphic modifications* option in the Ignore section.

15. Click *Finish* to complete the ECO import process.

The Cell Editor appears with the part information.

## Import ECO - ViewLogic(VL) Part

The Import ECO-ViewLogic (VL) option enables you to modify part data by using part data information available through ViewLogic parts.

## Steps

1. Choose *File – Import and Export*.

The Import and Export *wizard* appears.

2. Choose *Import ECO - ViewLogic(VL) Part* and click *Next*.

The Select Source page appears.

3. Select the input ViewLogic part.

The Select Package for ECO page appears.

4. Select the part that will be used to do the ECO and click *Next*.

The Select Destination for ECO page appears.

5. Select the library and the part to be modified and click *Next*.

The *ECO Messages* page appears. You can view the complete list of ECO modifications that are going to be done on the input cell. By default, Part Developer deletes the properties that are present in the cell but not in the input file and retains any symbol graphic modifications that have been done during ECO. You can turn off these default behaviors by using the *Property deletions* option and the *Graphic modifications* option in the Ignore section.

6. Click *Finish*.

The Cell Editor appears with the part information.

## Import ECO - Allegro Footprint

The *Import ECO - Allegro Footprint* option enables you to modify Design Entry HDL parts created from Allegro footprints.

## Steps

The steps are as follows:

1. Choose *File – Import and Export*.

The Import and Export wizard appears.

2. Select *Import ECO - Allegro Footprint* and click *Next*.

The Select Footprint page displays all of the footprints in `PSMPATH`.

3. Select the footprint you want to import and click *Next*.

The Select Destination page appears.

4. Enter the name of the cell to be created and the library in which it should be created and click *Next*.

The Preview of Import Data page appears. You can preview the data in *Logical Pins* and *Global Pins* grids.

5. Click *Next* to continue.

The ECO Messages page appears. You can view the complete list of ECO modifications that are going to be done on the input cell. By default, Part Developer deletes the properties that are present in the cell but not in the input file and retains any symbol graphic modifications that have been done during ECO. You can turn off these default behaviors by using the *Property deletions* option and the *Graphic modifications* option in the Ignore section.

6. Click *Finish* to complete the import process.

The Cell Editor appears with the part information.

## Import ECO - Die Text

The *Import ECO - Die Text* option enables you to modify parts created from die files.

### Steps

The steps are as follows:

1. Choose *File – Import and Export*.

The Import and Export wizard appears.

2. Choose *Import ECO - Die Text* and click *Next*.

The Select Source page appears.

3. Browse to locate the die file and click *Next*.

The Select Destination for ECO page appears.

4. Enter the name of the part to be modified, select the destination library, and click *Next*.

The Preview of Import Data page appears.

**5. Click *Next*.**

If there are duplicate pins in the die file, Duplicate Pin Resolver Dialog appears. You need to resolve the duplicate pins. For more information, see [Duplicate Pin Name Handling](#) on page 251.

**6. Click *Next*.**

The ECO Messages page appears. You can view the complete list of ECO modifications that are going to be done on the input cell. By default, Part Developer deletes the properties that are present in the cell but not in the input file and retains any symbol graphic modifications that have been done during ECO. You can turn off these default behaviors by using the *Property deletions* option and the *Graphic modifications* option in the Ignore section.

**7. Click *Finish* to complete the import process.**

The Cell Editor appears with the part information.

## Import ECO - DML Model

The Import ECO - DML Model option enables you to modify part data by using part data information available through DML models.

**Note:** If the cell on which the ECO is done has a single primitive and the ECO is done with an IBIS/DML model with multiple package or device and the *Group all Primitives* option is not selected, then the primitive name will remain unchanged but the contents of the primitive will be ECOed.

If the *Group all Primitives* option is selected, the existing primitive will be ECOed and the new primitives are added to the cell.

### Steps

**1. Choose *File – Import and Export*.**

The Import and Export *wizard* appears.

**2. Choose *Import ECO - DML Model* and click *Next*.**

The Select Source page appears.

**3. Select the input DML model.**

The Select Package for ECO page appears.

4. Select the library and part to be modified and click *Next*.
5. Select the device type that will be used to do the ECO and click *Next*.

The *ECO Messages* page appears. You can view the complete list of ECO modifications that are going to be done on the input cell. By default, Part Developer deletes the properties that are present in the cell but not in the input file and retains any symbol graphic modifications that have been done during ECO. You can turn off these default behaviors by using the *Property deletions* option and the *Graphic modifications* option in the Ignore section.

6. Click *Finish*.

The Cell Editor appears with the part information.

## Import ECO - IBIS Model

The Import ECO - IBIS Model option enables you to modify part data by using part data information available through IBIS models.

### Steps

1. Choose *File – Import and Export*.

The Import and Export *wizard* appears.

2. Choose *Import ECO - IBIS Model* and click *Next*.

The Select Source page appears.

3. Select the input IBIS model.

The Select Package for ECO page appears.

4. Select the library and part to be modified and click *Next*.

5. Select the device type that will be used to do the ECO and click *Next*.

The *ECO Messages* page appears. You can view the complete list of ECO modifications that are going to be done on the input cell. By default, Part Developer deletes the properties that are present in the cell but not in the input file and retains any symbol graphic modifications that have been done during ECO. You can turn off these default behaviors by using the *Property deletions* option and the *Graphic modifications* option in the Ignore section.

6. Click *Finish*.

The Cell Editor appears with the part information.

## Import ECO - Mentor Part

The Import ECO - Mentor Part option enables you to modify part data by using part data information available through Mentor parts.



Cadence strongly recommends you to use the Mentor to Design Entry HDL conversion on Windows platform with Mentor 8.9 release.

### Steps

1. Choose *File – Import and Export*.

The Import and Export *wizard* appears.

2. Choose *Import ECO - Mentor Part* and click *Next*.

The Select Source page appears.

3. Select the input Mentor part and click *Next*.

The Select Destination for ECO page appears.

4. Select the library and part to be modified and click *Next*.

The *ECO Messages* page appears. You can view the complete list of ECO modifications that are going to be done on the input cell. By default, Part Developer deletes the properties that are present in the cell but not in the input file and retains any symbol graphic modifications that have been done during ECO. You can turn off these default behaviors by using the *Property deletions* option and the *Graphic modifications* option in the Ignore section.

5. Click *Finish*.

The Cell Editor appears with the part information.

## Import ECO - Pin Grid

The Import ECO - Pin Grid option enables you to modify part data by using part data information available through pin information available in PDF datasheets.

## Steps

1. Choose *File – Import and Export*.  
The Import and Export wizard appears.
2. Choose *Import ECO - Pin Grid* and click *Next*.  
The Select Destination page appears.
3. Select the library and part to be modified and click *Next*.  
The Paste the data page appears.
4. Paste the pin grid data from the Clipboard and click *Next*.  
The Preview of the Derived Data page appears.
5. Verify the data and make changes if required. To continue, click *Next*.

The *ECO Messages* page appears. You can view the complete list of ECO modifications that are going to be done on the input cell. By default, Part Developer deletes the properties that are present in the cell but not in the input file and retains any symbol graphic modifications that have been done during ECO. You can turn off these default behaviors by using the *Property deletions* option and the *Graphic modifications* option in the Ignore section.

6. Click *Finish*.  
The Cell Editor appears with the part information.

## Import ECO - Pin Table

The Import ECO - Pin Table option enables you to modify part data by using pin information available in PDF datasheets.

## Steps

1. Choose *File – Import and Export*.  
The Import and Export wizard appears.
2. Choose *Import ECO - Pin Table* and click *Next*.  
The Select Destination page appears.
3. Select the library and part to be modified and click *Next*.

The Paste the data page appears.

4. Paste the pin grid data from the Clipboard and click *Next*.

The Preview of the Derived Data page appears.

5. Verify the data and make changes if required. To continue, click *Next*.

The ECO Messages page appears. You can view the complete list of ECO modifications that are going to be done on the input cell. By default, Part Developer deletes the properties that are present in the cell but not in the input file and retains any symbol graphic modifications that have been done during ECO. You can turn off these default behaviors by using the *Property deletions* option and the *Graphic modifications* option in the Ignore section.

6. Click *Finish*.

The Cell Editor appears with the part information.

## Export Capture Part (Windows Only)

The Design Entry HDL to Capture conversion is essentially the conversion of the symbol and chips view of a Design Entry HDL cell to a Capture part.

The following types of parts are converted:

- Normal flat parts
- Sized parts. Such parts are converted to homogeneous parts in Capture.
- Split parts. Such parts are converted to heterogeneous parts in Capture.
- Asymmetrical parts. Such parts are converted to heterogeneous parts in Capture.

**Note:** The translation does not support the creation of library-level symbols, such as title blocks and power symbols, into the Capture database.

**Note:** The Capture database supports a maximum of two symbols for each function in a part. Therefore, the user can choose a maximum of two symbols per group. Also, in order to have the full functionality of the part to be translated, the user must choose at least one symbol for each group. There is no equivalent of `FIXED_SIZE` symbols in Capture. Therefore, these symbols are not translatable.

## Steps

The steps are as follows:

**1.** Choose *File – Import and Export*.

The Import and Export wizard appears.

**2.** Choose *Export Capture Part (Windows Only)* and click *Next*.

The Select Source dialog box appears.

**3.** Select the part to be exported and click *Next*.

**4.** Select the package.

**5.** After you select the package, you need to select the symbol(s) from the list of symbols.

You can select a maximum of two symbols for conversion.

**Note:** Only those symbols that can be packaged into the selected package are displayed. Also, since Capture does not support the notion of symbols with fixed size, none of the Design Entry HDL symbols that have the `HAS_FIXED_SIZE` property will appear in the list of symbols.

**6.** If you want to use the symbol port names to represent the pins in Capture, select *Use Pin Name to write Capture Port Name*. Otherwise, the value of the `PIN_TEXT` property is used to represent the pin names in Capture. For example, a Design Entry HDL symbol has the `PIN_TEXT` property value as `ABC_CLOCK` for the symbol pin A. If you select the *Use Pin Name to write Capture Port Name* option, then the value of the Pin Name property in Capture for symbol pin A will be A. If the option is not selected, the value of the Pin Name property in Capture will be `ABC_CLOCK`.

**7.** Specify the Capture library in which to create the part and click *Finish*.

**Note:** In case the tool fails to find the Capture library in the specified location, it creates the new library automatically.

## Conversion Details

### Translation of the Chips (Physical) View

The following information is translated from the chips view of a Design Entry HDL part:

- Pin Names
- Package Properties
- Pin Types
- Part Aliases (equivalent packages)

### **Pin Names**

- The pin text on a Design Entry HDL symbol pin is translated into Capture as pin name. Alternatively, you can also select Design Entry HDL part's pin name instead of pin text.

**Note:** In the Design Entry HDL part, the pin text may not necessarily exist. In the symbols generated by Part Developer and Design Entry HDL prior to 14.0, the pin text on symbol pins was a plain text placed near the location of the pin. It is therefore possible that the automated translation may not be able to locate this pin text by position as pin text may not exist near the end of the pin stub. In such cases, the pin text may not be found or incorrectly linked to another text in the search region. If the pin text is not found, the tool automatically uses the pin names instead of pin text while naming the Capture pin. In 14.0, Design Entry HDL/Part Developer has a property on the symbol pin called `PIN_TEXT` that links in the exact pin text so as to avoid the ambiguity. It is recommended that the old symbols (created in releases prior to 14.0) are saved in Part Developer and verified before being converted to Capture. Part Developer associates the pin text to pin by moving the text into a `PIN_TEXT` pin property. If this is not done, then you may need to edit the translated Capture part to remove the texts that were placed as texts next to pins but did not get detected as pin texts as they were not as `PIN_TEXT` property.

- If the pin name is used instead of pin text, then the following characters are replaced:
  - `_GT_` to >
  - `_LT_` to <
- The sizable ports in the Design Entry HDL sizable part are converted to scalar ports of the converted homogeneous part.
- The vector bits are converted as scalar ports. For example, A[10] is converted to Capture pin A10. However, a vector bus is converted as is.

### **Package Properties**

- Design Entry HDL parts have the ability to support additional data on a pin such as pin loads. Such information is added to the symbol pins on Capture.
- The package properties are translated as follows:
  - `JEDEC_TYPE` to PCB footprint
  - `BODY_NAME` to Value as Capture Symbol property
  - `NC_PINS` to NC as Capture Symbol Property
  - `PHYS DES PREFIX` to Part Reference Prefix

## Part Developer User Guide

### Import and Export

---

#### ***Pin Type***

The pin types are converted as per the following table:

<b>Design Entry</b>	<b>HDL Pin Type</b>	<b>Capture Pin Types</b>
Input		INPUT
Output		OUTPUT
OC		Open Collector
OE		Open Emitter
TS		3-State
BIDIR		Bidirectional
POWER		POWER
UNSPEC		PASSIVE
GROUND		POWER
ANALOG		PASSIVE
OC_BIDIR		Bidirectional
OE_BIDR		Bidirectional
TS_BIDIR		Bidirectional
NC		Passive

#### ***Part Alias***

- The equivalent packages in a Design Entry HDL part (the list of packages in single primitive line in chips view) are made as aliases in the Capture part.
- The last character, `_`, is changed to `/` in the alias names in accordance with Capture standards for naming aliases.

#### **Translating Symbol (Logical) View**

The following information is translated from the Design Entry HDL symbol:

- Pin Names

- Pin Location
- Graphics - Symbol Shape
- Graphics - Pin Shapes
- Pin Properties
- Symbol Properties

### ***Pin Names***

See the section on pin names in translating from the chips view.

### ***Pin Locations***

Special care is taken to ensure that the location of pins is an exact match to that in the Design Entry HDL part. This allows the design translators to be based on graphical translation. If a pin is located within a bounding box (which is not in accordance with Capture standards), then it is converted as a zero-length pin and the pin shape is converted to graphic lines.

A Design Entry HDL part can have two types of pins in the body section: Power pins and NC pins. These are pins that are not located on the Design Entry HDL symbols.

For NC pins in the body section, the translation places a property on the symbol NC.

For Power pins in body section, the translation adds each of these pins on the bounding box of Capture starting from the top-left corner in such a way that no two pins on the part overlap at hotspot. These pins are added as invisible pins.

**Note:** If a translated part has pins within the bounding box and the symbol is edited in Capture, the Symbol Editor automatically moves the pins out to the bounding box.

### ***Graphics - Symbol Shapes***

The following graphics entities are translated:

- Line
- Rectangle
- Ellipse
- Arc

- Polyline
- Filled shapes

### ***Graphics - Pin shapes***

Design Entry HDL does not support pin shapes. The pin shapes in Design Entry HDL can consist of one or more graphics, such as line, arc, circle and so on. In translation to Capture, if a shape exists that matches to a Capture pin shape, then it is translated into an equivalent Capture shape.

The bubbled pins are translated as pin line shape in Capture. All pins that are on or inside the calculated bounding box are converted to zero-length pins.

### ***Pin Properties***

All pin properties are translated as is.

### ***Symbol Properties***

The following property names and case are modified:

- IMPLEMENTATION\_PATH to Implementation Path
- IMPLEMENTATION\_TYPE to Implementation Type
- IMPLEMENTATION to Implementation
- PSPICETEMPLATE to PSpiceTemplate
- NC\_PINS to NC. This symbol property is transferred from chips view.

## **Export EDAXML Part**

Using Part Developer, you can save the part information in EDAXML format. Essentially, the symbol and chips views of a Design Entry HDL part are saved in the XML format.

**Note:** Design Entry HDL supports multiple packages through the primitives in the chips.prt file. Each of these primitives maps to a unique XML part. The user needs to select the primitive that needs to be saved in the XML format.

The translation supports

- Normal flat parts.
- Sized parts. These are converted to single cells in XML.
- Split parts. These are converted to multiple cells in XML having the same source package.
- Asymmetrical parts. These are converted to multiple cells in XML having the same source package.

## Steps

The steps are as follows:

1. Choose *File – Import and Export*.  
The Import and Export dialog box appears.
2. Choose *Export EDAXML Part* and click *Next*.  
The Select Source page appears.
3. Select the Design Entry HDL library and the component and click *Next*.
4. Select the package to be saved in the XML format and click *Next*.
5. Specify the directory in which to save the XML file and click *Finish*.

**Note:** The XML file name is derived from the package name of the Design Entry HDL part.

## Conversion Details

When you save a part data in XML format, the symbol and chips information is written in the XML file.

### Translating Chips (Physical) View

The following information is translated from the Design Entry HDL part:

- Pin names
- Pin text
- Package properties
- Pin types

- Part aliases (equivalent packages)

### ***Pin Names***

The pin name on a Design Entry HDL symbol pin is translated into XML as pin name. The Design Entry HDL assertion character is translated as per the Capture assertion mechanism, where \ follows each character for low-asserted pins.

### ***Pin Text***

The pin text is added as the `PIN_TEXT` property to the ports in XML.

**Note:** The `PIN_TEXT` property may not necessarily exist in all Design Entry HDL parts. In the symbols generated by Part Developer and Design Entry HDL prior to the 14.0 release, the pin text on symbol pins was a plain text placed near the location of the pin. It is therefore possible that the automated translation may not be able to locate this pin text by position as the pin text may not exist near the end of the pin stub. In such cases, the pin text may not be found or incorrectly linked to another text in the search region. If the pin text is not found, the translation automatically uses the pin names instead of pin text while naming the XML pin. In the 14.0 release, Design Entry HDL and Part Developer have a property on the symbol pin called `PIN_TEXT` that links in the exact pin text so as to avoid ambiguity. It is recommended that the old symbols (created in releases prior to the 14.0 release) are saved in Part Developer and verified before converting them to XML. Part Developer links the pin text to pin by moving the text into a `PIN_TEXT` pin property. If this is not done, then you may need to edit the translated XML part to remove the texts that were placed as texts next to pins but did not get detected as pin texts as they were not as `PIN_TEXT` property.

### ***Package Properties***

`BODY_NAME` is deleted in the XML export. All other properties are added either as properties or under a special XML tag if the DTD has a special provision for the value of that property. An example of a property that goes under an XML tag is `REFDES_PREFIX`.

## Part Developer User Guide

### Import and Export

---

#### ***Pin Type***

The pin types are converted as per the following table:

Part Developer Pin Types	XML Direction	XML Types
INPUT	Input	Input
OC	Output	OC
OE	Output	OE
TS	Output	HIZ
OC_BIDIR	Bidirectional	OC
OE_BIDIR	Bidirectional	OE
POWER	Unspecified	POWER
NC	Unspecified	NC
ANALOG	Unspecified	ANALOG
UNSPEC	Unspecified	UNSPEC
TS_BIDIR	Bidirectional	HIZ
GROUND	Unspecified	POWER

#### ***Part Alias***

The aliases are added as a series of property PACKAGE\_ALIAS(*n*) , where *n* is the sequence number.

#### **Translating Symbol (Logical) View**

The following are translated from the symbol view:

- Pin Names
- Pin Location
- Graphics - Symbol Shape
- Graphics - Pin Shapes

- Pin Properties
- Symbol Properties

### ***Pin Names***

See Pin Names in the translation of the chips (physical) view for details.

### ***Pin Location***

Special care is taken to ensure that the location of pins is an exact match to that in the Design Entry HDL part. This allows the design translators to be based on graphical translation. If a pin is located within a bounding box then it is converted as a zero-length pin and the pin shape is converted to graphic lines.

A Design Entry HDL part has three type of pins in the body section of the `chips.prt` file: Power pins, NC pins, and GROUND pins. These are pins that are **not** located on Design Entry HDL symbols.

For NC pins in the body section, the translation places a property on the symbol called NC.

For Power pins in the body section, the translation adds each of these pins on the bounding box of the XML symbol starting from the top-left corner in such a way that no two pins on the part overlap at hotspot. These pins are added as invisible pins.

### ***Graphics - Symbol Shapes***

The following graphics entities are translated:

- Line
- Rectangle
- Ellipse
- Arc
- Polyline
- Filled shapes

### ***Graphics - Pin shapes***

Design Entry HDL does not support pin shapes. The pin shapes in Design Entry HDL can consist of one or more graphics, such as line, arc, circle and so on. While translating to XML, if a shape exists that matches to a Capture pin shape, then it is translated into an equivalent Capture shape, which is how the pins shape templates are represented in XML. For more details, see *OrCad Capture User's Guide*.

The bubbled pins are translated as pin line shape in XML.

### ***Pin Properties***

All pin properties are translated as is.

### ***Symbol Properties***

All pin properties are translated as is.

### ***Limitation***

The XML output from a Design Entry HDL part defaults the font information for all texts to the default in Capture.

## **Export Comma Separated Value (.csv) File**

Using Part Developer, you can save the part information in csv format.

**Note:** Part Developer will export symbol information even if there are no symbols created for the part. This is done on the basis of the number of slots for the part.

### **Steps**

The steps are as follows:

1. Choose *File – Import and Export*.

The Import and Export dialog box appears.

2. Choose *Export Comma Separated Value (.csv) file* and click *Next*.

The Select Source page appears.

3. Select the Design Entry HDL library and the component and click *Next*.
4. Select the package to be saved in the CSV format and click *Next*.
5. Specify the directory in which to save the CSV file and click *Finish*.

**Note:** The CSV file name is derived from the package name of the Design Entry HDL part.

## Export ViewLogic(VL) Part

Using Part Developer, you can save the part information in ViewLogic(VL) format.

### Steps

The steps are as follows:

1. Choose *File – Import and Export*.  
The Import and Export dialog box appears.
2. Choose *Export ViewLogic(VL) Part* and click *Next*.  
The Select Source page appears.
3. Select the Design Entry HDL library and the component and click *Next*.  
The *Select Associated Packages or Unassociated Symbols* page appears.
4. Select the package or symbol and specify the type of the ViewLogic part and click *Next*.  
The Select Destination page appears.
5. Specify the destination directory and click *Finish*.

## Export Mentor Part

Using Part Developer, you can save the part information in Mentor format.



Cadence strongly recommends you to use the Design Entry HDL to Mentor conversion on Windows platform with Mentor 8.10 release.

## Steps

The steps are as follows:

1. Choose *File – Import and Export*.

The Import and Export dialog box appears.

2. Choose *Export Mentor Part* and click *Next*.

The Select Source page appears.

3. Select the Design Entry HDL library and the component and click *Next*.

The Select Destination page appears.

4. Specify the destination directory and click *Finish*.

## **Part Developer User Guide**

### **Import and Export**

---

# **Verifying Parts**

---

After you complete the creation of a part, you can run various checks on it to ensure that they are correct.

To run checks:

1. Select *Tools – Verify*.
2. Select the required check.
3. Click *Verify*.

**Note:** In case you have a part that has names with the “(”, “)”, or “&” symbol, you should run the `hlibftb` library-checking utility from the command line with the `-advopt` option.

You can run the following checks:

- [View Verification](#)
- [Instantiation and Packaging](#)
- [Advanced View Checks](#)
- [VHDL Compilation](#)
- [Verilog Compilation](#)
- [Verify with Templates](#)

## **View Verification**

You can run the following checks:

- Symbol origin is centered.  
Checks whether the origin always lies within the symbol and the symbol is at a distance less than the maximum allowed offset from the origin.

- Tristate pins have input and output loads defined.  
Checks the presence of pin properties OUTPUT\_LOAD and INPUT\_LOAD for every tristate pin. This is denoted by the property OUTPUT\_TYPE =TS,TS.
- Mandatory properties present in package file.  
Checks whether the properties named BODY\_NAME, PART\_NAME, CLASS, and JEDEC TYPE are present in the chips.prt file.
- Consistent symbol name in symbol and package file.  
Checks whether the symbol text is the same as BODY\_NAME in the chips.prt file.
- Consistent symbol and package in the pin list.  
Checks whether the pins are the same across symbol and package views.

## Instantiation and Packaging

### con2con Instantiation and Packaging Checks

If you have the required license, selecting the *Instantiation and Packaging* option from the *Verify* dialog box runs the con2con instantiation and packaging checks—instead of hlibftb—by using the internal validation feature of Part Developer. The con2con checks help you quickly determine, if a part can be packaged. The following options are available:

- Use physical part rows for verification
- Verify front-to-back compatibility

Selecting the *Use physical part rows for verification* option enables checks between all the physical part rows associated with all the primitives in the part for front-to-back verification.

Selecting the *Verify front-to-back compatibility* option enables checks between the physical pin list of each primitive and that of the associated footprint.

### Running con2con Instantiation and Packaging Checks on a Library or a Cell

You can run instantiation and packaging checks on an entire library or on a particular cell.

To run the instantiation and packaging checks on an entire library, choose *Tools – Verify – Instantiation and Packaging (Verification of parts in system flow)* after opening a project file.

To run the instantiation and packaging checks on a cell, choose *Tools – Verify – Instantiation and Packaging (Verification of parts in system flow)* after opening the cell.

These checks can also be run from the command prompt. For more details, see [con2con](#) on page 370.

## Output Description

The `con2con` instantiation and packaging checks generate the following reports:

Filename	Contents
<code>&lt;project_name&gt;.log</code>	Errors and warnings for each cell
<code>&lt;project_name&gt;.rep</code>	Pass/fail report for each cell
<code>&lt;project_name&gt;.sum</code>	Numbers of cells verified, passed, and failed

## Running `hlibftb` with the PCB Librarian XL License

If you are running Part Developer using the PCB Librarian XL license and want to run the `hlibftb` utility, you need to set the value of the `cpm` directive `Instantiation_Packaging_Validation_Type` to 0.

If you are running Library Explorer or Library Flows using the PCB Librarian XL license and want to run the `hlibftb` utility, you need to set the environment variable `Instantiation_Packaging_Validation_Type` and assign the value 0.

## **hlibftb Instantiation and Packaging Checks**

In the PCB Librarian license, selecting the same option runs the `hlibftb` utility with the following options:

- Use Project ptf files for verification
- Use allegro board (netrev)

If you do not select the *Use Project ptf files for verification* option and the checks are run, each symbol is instantiated on the schematic sheet with only the symbol properties present on the symbols which is then packaged.

If *Upto PCB Editor Board* is checked, Part Developer looks for the `JEDEC_TYPE` property in the `chips.prt` file as the instances on the schematic sheet do not have any

JEDEC\_TYPE property. When update allegro board is run, it looks for the JEDEC\_TYPE value (the one in the `chips.prt` file) in the PSMPATH and if that footprint is not available in the PSMPATH, then it gives an error similar to the following example:

```
Symbol 'DIP3' for device 'ZENER_SOT23_A2C3' not found in PSMPATH or must be  
"dbdoctor"ed.
```

If the *Use Project ptf files for verification* is selected, Part Developer instantiates each symbol on the schematic sheet with the part table properties in the cell-level part-table file annotated onto the instances. This design is then packaged and update allegro board is run. If the JEDEC\_TYPE property is one of the key or injected properties in the ptf file, then on running update allegro board, the JEDEC\_TYPE property on the instance gets the precedence and if this value is there in the PSMPATH, it will update the board successfully.

- Generate Pass/Fail report

This option is enabled only if you select more than one part or when you select a library that has more than one part.

**Note:** If you select the Generate Pass/Fail option, each part is verified separately. This is a time-consuming process.

## Advanced View Checks

Select this option to launch Rules Checker. You can run your own custom-defined checks using Rules Checker.

## VHDL Compilation

Use this option to compile the generated VHDL wrapper. You can use either NCVHDL or CV to compile the wrapper. You can specify the tool to compile the VHDL wrapper in the *Enter command* in the VHDL compilation dialog box. This dialog box is displayed when you click the *Options* button in the Verifications dialog box.

## Verilog Compilation

Use this option to compile the generated Verilog wrapper. You can use NCVERILOG to compile the wrapper. You can specify the tool to compile the Verilog wrapper in the *Enter command for Verilog compilation* dialog box. This dialog box is displayed when you click the *Options* button in the Verifications dialog box.

## Verify with Templates

Select this option to verify a selected part against a .tpl template. The result of the verification is displayed in a report. The verification is done as per the following rules:

### Property Checks

The property check is done on all packages for the following:

- All properties listed in the template for a package must exist in each package of the part.
- The value of the property in each package must match the value in the template unless the value in template is "?" or blank.

### Pin Load Checks

This is done on all pins in all packages as per the following rules:

- If PINUSE="UNSPEC" exists for a pin, all checks are bypassed on that pin.
- If a pin type is not determined, it is an error.
- If a pin type is determined, its load value is checked against the load value of that pin type in the template. An error is generated if the load values do not match.
- Error is shown if any of the loads for a pin type is missing.

### Symbol Checks

All symbols are checked for a given part as per the following rules:

- All symbols must have at least one connection with a line stub or a bubble else an error stating that check cannot proceed is shown.
- All lines are assumed to be vertical or horizontal. Arcs are not supported in this release.
- Each bubble is interpreted to have two virtual stub lines - horizontal and vertical.
- No two connections can have the same X, Y coordinates else error is shown to the user.
- Location of connections is derived based on the direction of the stubs attached to the connection. Therefore, only connections that have a stub or a bubble on them are checked.
- Stub length is calculated based on the integer average of all stub lengths.

- The outline of the body is derived by searching for the perpendicular line from the end of stub. As the stub size varies, the search is made within the range of the stub size variance. After getting a single outline, the rest are traced as the ones connected to the outline end points. The procedure is executed recursively for each detected outline.
- Minimum pin spacing are calculated for all sides (Top, Left, Right, Bottom).
- For pin texts, the property `PIN_TEXT` is used. If it is not found, pin texts are searched for within 1/3 of the average stub length from the end of the stub for each connection. In addition, the location (x,y) of the pin note must not be mis-aligned by more than 1/2 pin spacing.
- Grid is derived by taking the highest common factor of all differences of values of X and Y on respective coordinates. Only the connection (Logic) grid is derived. The symbol grid is not derived even though the template mentions it as Symbol Grid.
- All properties listed in the template for symbols must exist on each symbol in the part.
- The value of the property in each symbol must match the value in the template unless the value in template is "?" or blank.
- The alignment of the property must match the one specified in the template.
- The visibility of the property must match the one specified in the template.

### **Pin Checks**

Each pin is checked as per the following rules:

- Each pin based on the type as defined in the template must be at the location area in symbol as defined in the template for that type.
- The text size of the pin must match the size specified in the template.
- *Use Pin Names For Text* is checked only if the template sets it to true.
- The text style for pin text is checked with value in template. If the style is vertical for top and bottom pins and horizontal for pins on left and right, then the style is considered to be Automatic. The angles of 90 and 270 are considered equivalent and vertical and 0 and 180 are considered equivalent and horizontal.
- All pins with spacing less than the spacing specified in the template are marked as errors.

### **Grid Checks**

Grid checks are done with the following rules:

- Conversions are done for calculating and matching the grid values for Inches, Metric, and Fractional (Fractional is currently not supported in Part Developer). This is then matched with the template value.

### **Outline Checks**

The outline checks are done with the following rules:

- All detected outlines are checked to match the thickness specified in the template.

### **Minimum Size Checks**

- The minimum pin spacing values on the left and right is read for each symbol and verified against the minimum pin spacing left and right value stored in the template. An error is generated if the value in the symbol is less than that of the value stored in the template.
- The minimum pin spacing values on the top and bottom is read for each symbol and verified against the minimum pin spacing top and bottom value stored in the template. An error is generated if the value in the symbol is less than that of the value stored in the template.
- The minimum symbol height value is read for each symbol and verified against the minimum symbol height stored in the template. An error is generated if the value in the symbol is less than that of the value stored in the template.
- The minimum symbol height width is read for each symbol and verified against the minimum symbol width stored in the template. An error is generated if the value in the symbol is less than that of the value stored in the template.

The output of the verification is displayed in a dialog box. The output is divided into two sections, Overview and Details.

In the Overview section, the overview of the differences are displayed. In the Details section, the differences are detailed.

## **Part Developer User Guide**

### Verifying Parts

---

---

# Interface Comparator

---

Interface Comparator is a mechanism by which you can compare the pin lists of two interfaces, such as a package and a symbol, identify the differences, and then update the pin list of one of the objects to match the other or both with respect to each other.

The following scenario explains the need and usefulness of this feature. Suppose you create a part where you develop the packages and symbols separately. After creating them, you realize that there are some symbols that are unassociated with any of the packages. Now, such a part is not usable in the design flow because the symbols must be packageable into one of the packages. So, to make a symbol packageable, you need to update the pin list of the symbol to match the pin list of the package or vice-versa. For large pin-count parts, this task, when done manually, can be tiresome and error-prone. The Interface Comparator feature enables you to automatically identify the differences between the selected symbol and the package and update the pin list of either the symbol or package to match the other.

Using Interface Comparator, you can compare:

- a function group and a symbol.
- two function groups of a package.
- two symbols.

After the comparison, you can choose to update the pin list of one of the compared objects with respect to the pin list of the other or both with respect to each other.

## Steps to Run Interface Comparator

1. Right-click on a package, a function group, or a symbol in the Cell Editor tree.
2. From the menu, select *Interface Comparator*.
3. Select the first object. The first object drop-down list displays all the function groups across all the packages of the part and all the symbols.
4. Select the second object. The drop-down list displays all the function groups and the symbols that have a pin list that is different from the first object.

**5.** Click *Compare*. This comparison is only based on the logical pin list.

Part Developer runs a comparison check on the two objects and displays the pins in the following way:

- The pins that are present only in the first object are displayed in the *Only in First* list box.
- The pins that are present only in the second object are displayed in the *Only in Second* list box.
- The pins that are common to both the objects are displayed in the *Common* list box.

Next, you need to determine how to synchronize the pin lists of the two objects.

Synchronization is the process by which the pin list of one of the objects is updated to

match the pin list of the other object. This is done by either adding or removing pins.

Synchronization can be done in one of the following ways:

- First with Second

In this method, the second object is treated as the master object and the pin list of the first object is updated to match the pin list of the second object. Extra pins in the first object are deleted from the first object. If the second object has additional pins, these pins are added to the first object.

- Second with First

In this method, the first object is the master object and the pin list of the second object is updated to match the pin list of the first object. The pins that are present in the first object but not in the second object are added to the second object. Pins that are present in the second object but not in the first are deleted from the second object.

- Both

In this method, both objects are treated at par. Only the common pins are retained in both the objects and the extra pins, if any, are deleted.

**6.** Select the synchronization method.

The pin list of the objects are updated automatically by Part Developer.

### **Points to Remember when Running Interface Comparator**

- In case logical pins are added to a package after synchronization, you need to open the package in the Package Editor and manually specify the logical-to-physical pin mapping for the new pins. An example part, MYPART, is used to explain this.

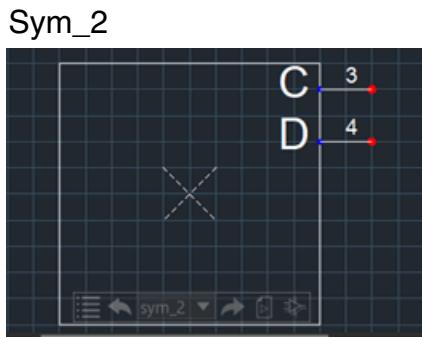
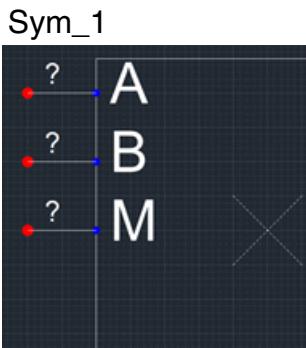
## Part Developer User Guide

### Interface Comparator

MYPART has one package MYPART\_DIP with four pins, A, B, C, and D. These four pins are split into two function groups, with pins A and B forming the first function group and pins C and D forming the other. The pin list of the package is displayed below:

	Name	Type	S1	S2
1	A	INPUT	1	-
2	B	INPUT	2	-
3	C	OUTPUT	-	3
4	D	OUTPUT	-	4

MYPART also has two symbols, sym\_1 and sym\_2. The symbols are displayed below:



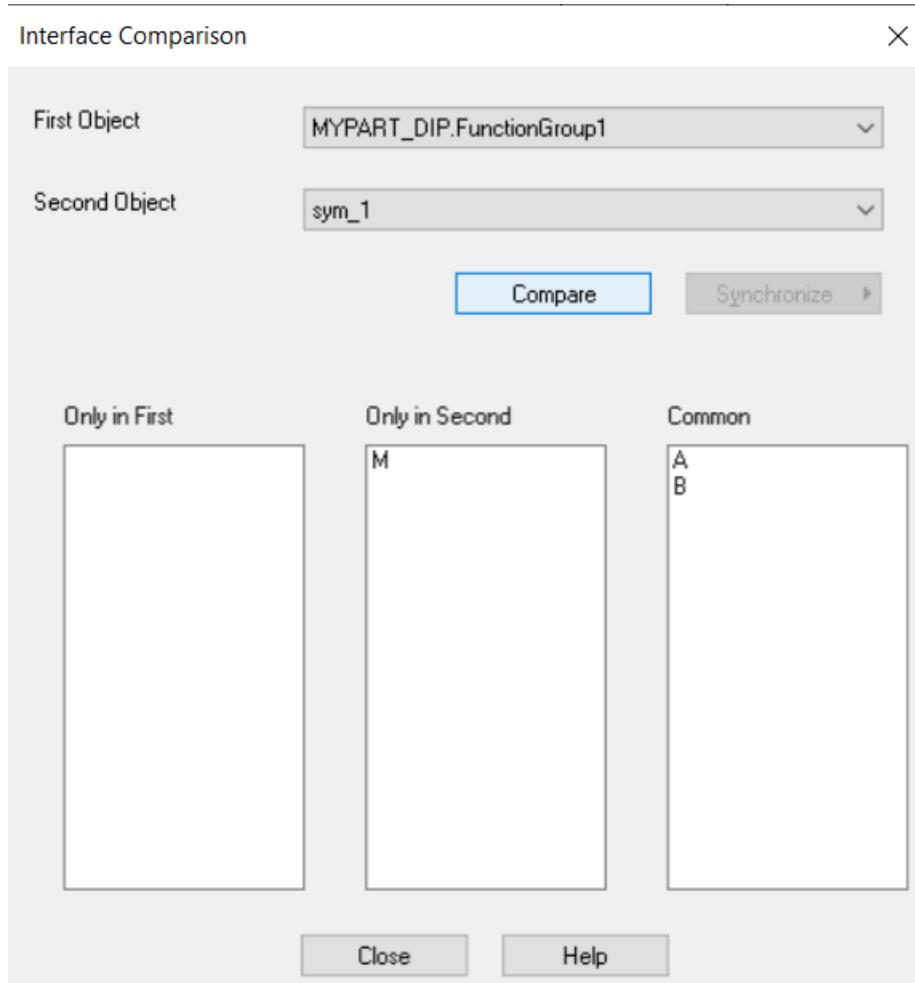
Sym\_2 is associated with the second function group. Sym\_1 has pins A and B and an extra pin M. This results in sym\_1 being not associated with any function group.

If Interface Comparator is used to synchronize the function group with the symbol, the logical pin M will be added to the first function group. Part Developer ensures that the synchronization does not destroy the existing function group-to-symbol associations, for example, the association of sym\_2 with function group 2.

## Part Developer User Guide

### Interface Comparator

When Interface Comparison is run, the Interface Comparator dialog box displays the following:



Now, when you synchronize with the *First with Second* option, pin M is added to the first function group. In the Logical Pins grid, pin M appears with a hyphen in the S2 column, thus showing that it is not present in the second function group. As shown in the graphic below, you will need to specify the physical pin numbers for the new logical pin, M, and do the mapping.

### MYPART\_DIP After Synchronization

	Name	Type	S1	S2
1	A	INPUT	1	-
2	B	INPUT	2	-
3	C	OUTPUT	-	3
4	D	OUTPUT	-	4
5	M	UNSPEC	-	-

Pin M added after synchronization

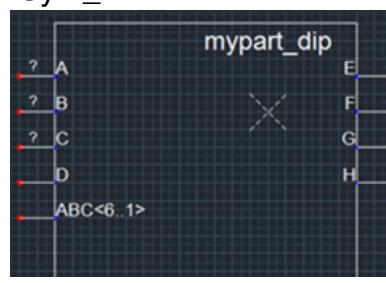
- When a pin is added to the symbol, the location for the pin is read from the setup options.
- When a pin is added to a package, attributes such as pin type, pin location, loading values, and checks are taken from the global data. Therefore, if the package, function group, or symbol from which the pin is added has attribute values that are different from the global data, those values are ignored and the attribute values in the global data are used.
- If pins are added to a symbol, Part Developer attempts to fit in as many pins as possible in the existing symbol outline. If the pins do not fit into the existing symbol outline, Part Developer automatically extends the symbol outline. In case the symbol outline is extended and the *Preserve Pin Position* option is checked on the Symbol Pins page of the Symbol Editor, Interface Comparator keeps the positions of the existing pins intact when adding the new pins.
- If some bits of a vector pin are deleted after synchronization, Part Developer displays the vector pin in an expanded format with the remaining bits.

For example, consider the following two symbols. Sym\_1 has a vector pin ABC with 6 bits, and sym\_2 has a vector pin ABC with 8 bits.

Sym\_2



Sym\_1



## Part Developer User Guide

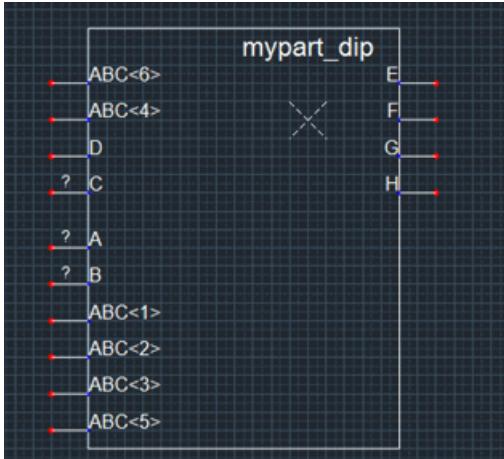
### Interface Comparator

On running Interface Comparison on the two symbols, the Interface Comparator dialog box appears as follows:



Now, when you synchronize with the *First with Second* option, the bits 7 and 8 are deleted from sym\_2. As shown in the graphic below, the vector pin ABC appears in sym\_2 with all the six bits expanded.

Sym\_2 After Synchronization



- After comparing a function group with another, it may be possible that a pin has hyphens in all the slots. In such a case, the pin is deleted from the package.

**Part Developer User Guide**  
Interface Comparator

---

---

## **SI Model Interface Comparison**

---

The SI Model Interface Comparison is a mechanism by which you can compare between pin name, number, type, and model name of DML model, IBIS model, Allegro footprint, and a schematic part.

The intent of this comparison is to enable you to identify or verify the compatibility between two models. Depending on the result, you may want to update the model or select another model for use in the design flow.

**Note:** The translate.cpm file directives are not applied when SI Model comparisons are run.

### **To run the SI Model Interface Comparison:**

1. Select *Tools – SI Model Interface Comparison*.

The SI Model Interface Comparison dialog box appears.

Now, you need to select the first model.

2. Select the model type from the *Select Type* drop-down list.
3. Select the model in the *Choose Model* field. You can browse and select the file that contains the selected model. For example, if you selected Schematic Part (Chips) in the Select Type field, then you need to select a library part. The list of available packages or devices are displayed below the Choose Model list box. You need to select one of the devices/packages from the list.
4. To select the second model, repeat steps and 3 in the Second Model group box.
5. Click *Compare* to compare between the two models.

### **SI Model Interface Comparison Results**

The SI Model Interface Comparison Results displays the results of the comparison. For more information, see [SI Model Interface Comparison Results](#) on page 449.

## **Part Developer User Guide**

### SI Model Interface Comparison

---

---

## **Part Logging and Versioning**

---

Part Logging and Versioning is the feature by which you can store and view the following information about a part:

- a log of all actions done on a part, such as symbol creation and pin modifications.
- major and minor revision numbers of all the views.

By default, whenever you start part logging, Part Developer puts the major revision number for all views and the cell as 1. The minor revision number is 0. Then, depending upon the type of changes you make to the part or views, the major or minor numbers of the part or views are updated automatically by Part Developer.

**Note:** Whenever a major or minor revision number of a view is incremented, the major or minor revision number of the cell is also incremented. For example, a part and its package and symbol views have a major revision number 1 and a minor revision number 0. Now, the package undergoes a modification, resulting in a major revision number change for the package. This will result in the major revision number of the part to be incremented as well. So, the part and the package will have the major revision number as 2. Now, the symbol is modified, resulting in the major revision number of the symbol to be incremented. This will also result in the major revision number of the part to be incremented. So, the part will now have the major revision number as 3 while the major revision numbers of the package and symbol continue to be 2. See [Modifications that Result in Major and Minor Number Changes](#) on page 337 for details.

The part log and version information is stored in the metadata view of the part. The metadata view contains the following files:

- master.tag

The `master.tag` file is the control file that ensures that Part Developer treats this directory as a valid view.

- revision.dat

The `revision.dat` file stores the version information for a part. The information stored are the name of the view, the type of the view, major and minor revision

## Part Developer User Guide

### Part Logging and Versioning

numbers, the date and time of creation, the name of the creator, the library to which the part belongs, and the modification history.

#### revision.log

The `revision.log` file logs all the activities that are done on the part. The information stored are the time of the modification, the name of the modifier, the type of modification, and a detailed description of the type of change.

Both `revision.dat` and `revision.log` files are written when the part is saved.

#### pinlist.txt

The `pinlist.txt` file stores the list of pins that have been added to the part.

Part Developer will start logging all the changes that you make on the part and also automatically increment the major and minor versions of the views and the part as required.



***Do not manually edit the `revision.dat` and `revision.log` files outside of Part Developer. If these files are manually edited, Part Developer may fail to give you the correct version and logging information for the part.***

## Revision Editor

The Revision Editor provides the view to the part logging and version information.

Revision								
	Name	Type	Status		Revision		Creation	
			Major	Error	Major	Minor	Date/Time	User
1	mypart	cell	Created	Yes	1	0	05/06/03,06:33:59	aoyon
2	sym_2	symbol	Created	No	1	0	05/06/03,07:08:59	aoyon
3	sym_1	symbol	Created	No	1	0	05/06/03,06:33:59	aoyon
4	MYPART	package	Created	Yes	1	0	05/06/03,06:33:59	aoyon
5	MYPART_1	package	Created	No	1	0	05/06/03,06:33:59	aoyon

It has the following fields:

### **BaseLine on Save**

Starts or stops part logging and versioning.

### **Name**

Shows the name of the cell and its views.

### **Status - Major**

Shows the major status for the part. The possible values are:

- **Created**  
This appears whenever a new cell or a new view is created.
- **Baselined**  
This appears when the part logging is started the first time or restarted.
- **Modified**  
This appears whenever the *BaseLine on Save* option is deselected and modifications are made to the part.

### **Error**

Identifies the view that has errors. By default, if any of the views has an error, the cell will always appear with a *Yes* status in the *Error* column.

### **Revision-Major**

Displays the major revision number for the cell or the view.

### **Revision-Minor**

Displays the minor revision number for the cell or the view.

### **Creation - Date/Time**

Displays the creation date and time.

### **Creation - User**

Displays the login information of the creator.

### **Creation - Path**

Displays the library and cell names in a `<libname>.<cellname>` notation.

### **LastModification - Date/Time**

Displays the date and time when the view was modified.

### **LastModification - User**

Displays the login information of the user who has modified the part or the view.

### **LastModification - Path**

Displays the library and cell names in a `<libname>.<cellname>` notation.

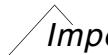
## **Starting Part Logging and Versioning**

To start part logging and versioning:

1. Load the part in the Cell Editor.
2. Select the *BaseLine on Save* option.
3. Save the part.
4. To view the change log and revision numbers in Part Developer, close the part and reload it.

The Revision Editor will appear with the major version showing *Baselined* for the cell and all its views. Now, whenever any change is made to the cell, the revision number will get updated as required.

**Note:** The changes are not dynamically reflected in the Revision Editor. You will need to reload the part to see all the revision history.

 *Important*

The *BaseLine on Save* option is deselected on reload of the part. You will need to select the *BaseLine on Save* option before saving the part to ensure that the revision numbers are updated.

In case a baselined part is saved without the *BaseLine on Save* option unchecked, then a `.baselined` file is created in all the views. For example, `chips.prt.baselined` is created in the chips view. Now, if the part is saved with the *BaseLine on Save* option checked, then the views are compared against the `.baselined` files, and revision numbers are updated accordingly.

 *Important*

Part logging is possible only for error-free parts.

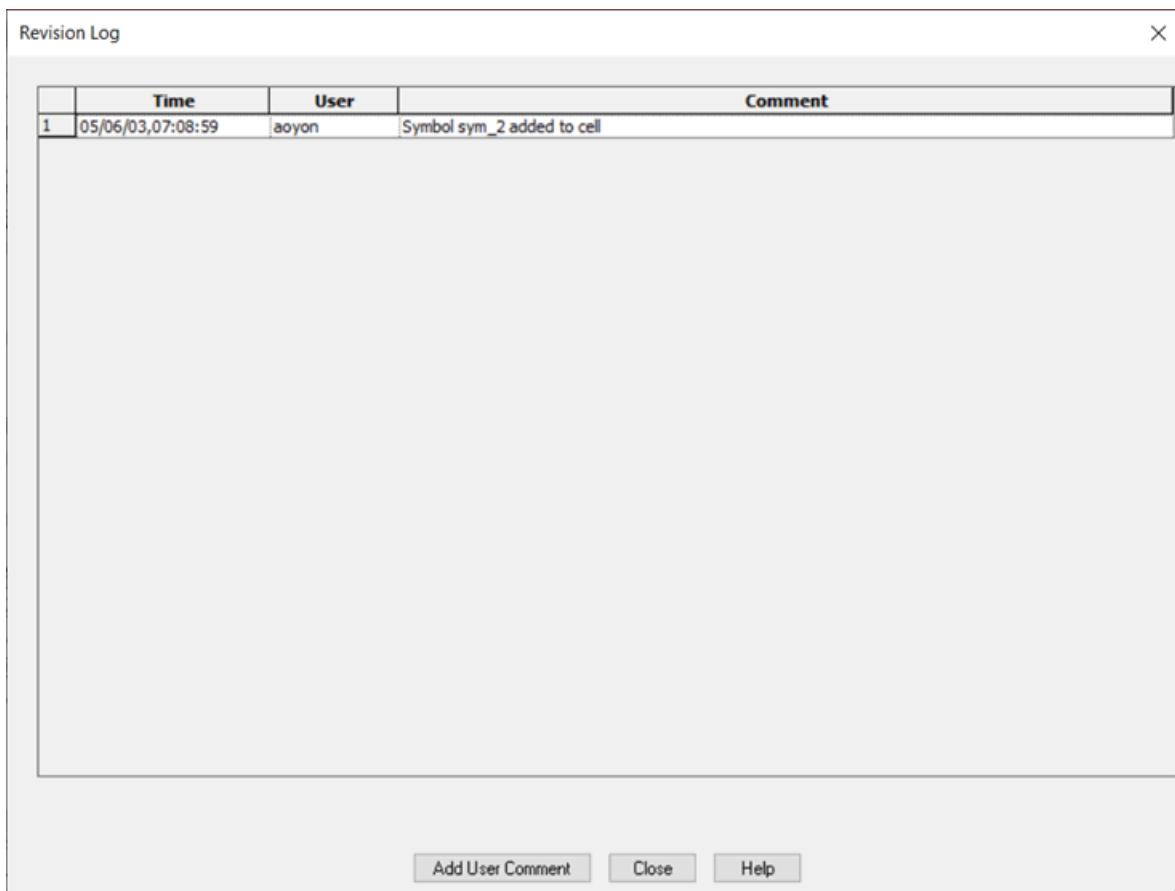
## Viewing the Revision Log

1. To view the revision log, click *Change log*.

The Revision Log dialog box appears displaying the log of changes made to the part and its views.

## Part Developer User Guide

### Part Logging and Versioning



## Adding Your Comments to the Revision Log

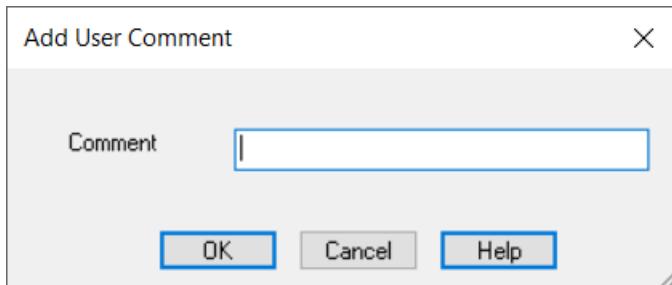
Part Developer allows you to write to the revision log of a part. To do so:

1. Click *Change Log*.

The Revision Log dialog box appears.

2. Click *Add User Comment*.

The Add User Comment dialog box appears.



3. Enter the comment that you want to add to the log file and click *OK*.

The comment gets written to the `revision.log` and appears in the Revision Log dialog box.

Revision Log			
	Time	User	Comment
1	05/06/03, 07:08:59	aoyon	Symbol sym_2 added to cell
2	08/22/23, 10:11:04		my first comment

## Stopping Part Logging and Versioning

- To stop part logging and versioning, deselect the *BaseLine on Save* option.

## Restarting the Part Logging and Versioning

- To start part logging and versioning, select the *BaseLine on Save* option.

## Modifications that Result in Major and Minor Number Changes

The changes that impact the design flow are considered major changes, leading to an increment in the major revision number. All other changes are treated as minor changes and result in incrementing of the minor revision number.

## Modifications that Result in a Major Number Change for a Part or a View

The following modifications cause a change in the *Major Number*:

1. Addition, deletion, or renaming of any view
2. Modification of the JEDEC\_TYPE property
3. Addition or deletion of pins from packages and symbols
4. Modification of pin-type for an existing pin
5. Renaming of pins in the package or symbols
6. Addition, deletion, renaming of symbol text
7. Modification of a symbol pin position
8. Addition or deletion of slots from a package
9. Change in the low-assertion character setup for a part, resulting in change of how the low-assertion character is stored in the chips or symbol view
10. Change in the distribution of pins among slots
11. Change in mapping information of existing pins
12. Global renaming of pins
13. Global deletion of pins
14. Addition or deletion of a model for a wrapper
15. Modification of a model for an existing wrapper
16. Mapping or unmapping of pin to m-port mapping in the mapfile(wrapper)
17. Addition, deletion, or modification of binding statements for a VHDL wrapper/mapfile
18. Addition or deletion of pin to model port mapping for primitives from Verilog/VHDL mapfiles
19. Modification of the default\_model property for a primitive in the map view
20. Addition, deletion, or modification of a model for a primitive entry in the mapfile
21. Annotation of parameters or generics onto symbols
22. Deletion of annotated parameters or generics from the symbols
23. Updating of pin-mode/port-type of pins during Verilog/VHDL mapfile(wrapper) creation
24. Modification of the UPPERCASE property of primitives

**Modifications that Result in a Minor Number Change for a Part or a View**

1. Addition, deletion, or renaming of an additional package property
2. Addition, deletion, or renaming of a package pin property
3. Addition, deletion, or renaming of a package alias
4. Modification of the electrical class of a package
5. Modification of the reference designator prefix of a package
6. Addition, deletion, or renaming of a symbol property
7. Addition, deletion, or renaming of a symbol pin property
8. Modification of the symbol outline (thickness/width/height)
9. Modification of the color of symbol outline
10. Change of pin-shape
11. Modification of symbol and symbol pin attributes
12. Addition or deletion of a model alias from the primitive in the mapfile

## **Part Developer User Guide**

### Part Logging and Versioning

---

## **Advanced Tasks**

---

This chapter describes the advanced configuration tasks to get the most out of Part Developer.



### **Caution**

***Any mistake when performing tasks described in this chapter may cause incorrect or unknown behavior. Therefore, the tasks should be performed with extreme caution and preferably under guidance of the Cadence Customer Support team.***



### **Tip**

All the configuration tasks should be done on the configuration files at the CDS\_SITE location.

## **Creating Personal Configuration Files through the CDS\_SITE Environment Variable**

If you have a network installation of Cadence tools, you will have more than one user accessing the central configuration files, such as `propfile.prop`. Because different users may have different needs, it is not always possible to modify the installed configuration files.

You can overcome this problem by creating your own copies of the configuration files and then creating a CDS\_SITE environment variable that points to this location. The CDS\_SITE environment variable ensures that Part Developer looks at the property files stored at the location pointed to by the CDS\_SITE variable.

To set up the CDS\_SITE environment variable:

1. Create a directory with any name, say `d:\local_pdv`.
2. Now in the `local_pdv` directory, create the directory structure `cdssetup\LMAN`.

So the directory structure should appear as follows:

d:\local\_pdv\cdssetup\LMAN

- 3.** Copy the configuration files from <your\_inst\_dir>\share\cdssetup\LMAN to d:\local\_pdv\cdssetup\LMAN

The configuration files that can be copied are:

- propfile.prop
- cds.cpm from <your\_inst\_dir>\share\cdssetup\projmgr

**Note:** Instead of copying the entire installation project file (cds.cpm), you can create your own site project file (site.cpm) containing only site-specific configuration.

- translate.cpm
- MainFrameMenu.panel
- CellEditorMenu.panel
- setup.cpm

## Cleaning the Part Developer Registry Entry

By default, Part Developer remembers the path to the project files and the window sizes by writing them down to the registry. To start Part Developer afresh, give the following command on the command prompt:

```
pdv -cleanreg
```

This cleans the registry entries made for Part Developer and then launches Part Developer.

## Configuring to Accept ? as the Only Placeholder Property Value

By default, Part Developer is configured to accept any character as the placeholder property value. This behavior is determined by the `Symbol_ValidateSoftPropValue` directive in the <project>.cpm file.

To ensure that only ? can be used as the placeholder property value:

- 1.** Open the <project>.cpm file.
- 2.** Go to the START\_PDV section.

3. Change the value of the `Symbol_ValidateSoftPropValue` directive to '1'.
4. Save the file and restart Part Developer.

This new configuration will be valid for all new parts, with Part Developer reporting errors for parts that do not have ? as the placeholder property values.

To make this change valid for all new projects, make the change in the `cds.cpm` file at `<CDS_SITE>\share\cdssetup\projmgr`.

## Translating Pin Types on Copy-Pasting from PDFs and Importing Data from CSV Files

The automatic translation of pin types is controlled through a file called `propfile.prop`. This file is located at `<CDS_SITE>\share\cdssetup\LMAN`.

The entry in the `propfile.prop` file that controls the automatic conversion is as follows:

```
(PackagePinType
  (PreDefined
    "ANALOG,BIDIR,GROUND,INPUT,OC,OC_BIDIR,NC,OE,OE_BIDIR,OUTPUT,POWER,TS,TS_BID
     IR,UNSPEC")
  (Translation
    (INPUT-OUTPUT "BIDIR")
    (INPUT\OUTPUT "BIDIR")
    (INPUT/OUTPUT "BIDIR")
  )
)
```

The Translation entry ensures that whenever a pin type of INPUT-OUTPUT, INPUT\OUTPUT, or INPUT/OUTPUT is encountered, it is always converted to BIDIR. If required, the pin type can be changed. However, you need to ensure that the pin type should be mentioned in uppercase. For example, if you want the pins to be converted to Input pin type, the Translation entry should be made in the following way:

```
(PackagePinType
  (PreDefined
    "ANALOG,BIDIR,GROUND,INPUT,OC,OC_BIDIR,NC,OE,OE_BIDIR,OUTPUT,POWER,TS,TS_BID
     IR,UNSPEC")
  (Translation
    (INPUT-OUTPUT "INPUT")
  )
)
```

```
(INPUT\OUTPUT "INPUT")
(INPUT/OUTPUT "INPUT")
)
)
```

**Note:** You can add your own translation entries. However, the translated pin type must be one of the predefined pin types.

## Specifying Default Pin Types for Text Import

You can specify default pin types for pins to be imported from text files in the `setup.cpm` file.

The `Import_Text_DefaultPinType 'UNSPEC'` directive is used to set the default pin type for pins for which a pin type could not be generated after pin type translation during text import.

## Specifying Default Pin Types for Allegro Footprint Import

You can specify default pin types for pins to be imported from Allegro footprints in the `setup.cpm` file.

The `Import_AllegroFtprint_DefaultPinType 'BIDIR'` directive determines the default pin type of pins imported through Allegro footprint import.

## Adding External Tools to Part Developer

Part Developer menus can be modified to add your own menu items. If you are using the PCB Librarian XL license, this is done by modifying the following panel files:

- `MainFrameMenu.panel`
- `CellEditorMenu.panel`
- `CellEditorMenu_SymEditor.panel`

For any other license, only the first two files need to be modified.

Copy these panel files from `<CDS_ROOT>\share\cdssetup\LMAN` to `<CDS_SITE>\cdssetup\LMAN` and modify them to control the appearance of the menu items.



**Caution**

***In case you make the changes only to the MainFrameMenu.panel file, the changes will be visible up to the point when a project is loaded in Part Developer. The changes in the menu will not be visible when a part is loaded. Similarly, if the changes are made only in the CellEditorMenu.panel file, the new menu items are visible only when a part is loaded. In case you want the menu items to appear at all times, make the necessary changes to both the panel files. If you are using the PCB Librarian XL license, make sure that the changes you make in the CellEditorMenu.panel file are also made in the CellEditorMenu\_SymEditor.panel file.***

To add a new menu item:

1. Launch a text editor of your choice and open the panel files.
2. Add the following lines to the panel files:

```
(Component MENU (attributes "<program name>,true,false,1,false") (Command  
    LU_Mnu_Execute_<command>) (Arguments "<name of exe> [%?% -proj %PROJECT% -  
    cdslib %CDSLIB% -lib %LIB% -cell %CELL%]"))
```

where:

- <program name> is the name of the new menu item.
- The value for Command should always be prefixed by LU\_Mnu\_Execute. If there are more than one user-defined menus, Command should have a different suffix for each, such as LU\_Mnu\_Execute\_Check and LU\_Mnu\_Execute\_Write .
- Arguments are similar to the command lines, i.e. name of the executable followed by arguments.
- In case you are adding a Cadence tool, then %?% displays a dialog box with the arguments. This provides the ability to change the arguments at runtime.
- %PROJECT% is substituted by the current project path.
- %CDSLIB% is substituted by the current cdslib path.
- %LIB% is substitute by the current library.
- %Cell% is substituted by the current cell.

## Example 1

Add a menu item to launch *Write* from the *Tools* menu of Part Developer.

### Steps:

1. Open `MainFrameMenu.panel` and the `CellEditorMenu.panel` files in a text editor. If you are using the PCB Librarian XL license, open the `CellEditorMenu_SymEditor.panel` file also.
2. Go to the `(Component MENU (attributes"&Tools,true,false,1,true")` line. This section describes the items of the *Tools* menu.
3. Go to the end of this section and add the following lines:

```
(Component MENU  
    (attributes"Write,true,false,1,false")  
    (Command LU_Mnu_Execute_Write)  
    (Arguments "write.exe")  
)
```

4. Save the file and launch Part Developer.
5. Click *Tools*.

*Write* should be the last menu item.

## Example 2

Add a menu item to launch CheckPlus from the *Tools* menu of Part Developer.

### Steps:

1. Open `MainFrameMenu.panel` and `CellEditorMenu.panel` file in a text editor. If you are using the PCB Librarian XL license, open the `CellEditorMenu_SymEditor.panel` file also.
2. Go to the `(Component MENU (attributes"&Tools,true,false,1,true")` line. This section describes the items of the *Tools* menu.
3. Go to the end of this section and add the following lines:

## Part Developer User Guide

### Advanced Tasks

---

```
(Component MENU (attributes "CheckPlus,true,false,1,false") (Command  
    LU_Mnu_Execute_Check) (Arguments "checkplusUI.exe %?% -proj %PROJECT% -cdslib  
    %CDSLIB% -lib %LIB% -cell %CELL%") ) I
```

4. Save the file and launch Part Developer.
5. Load a part in Part Developer.
6. Click *Tools*.

*CheckPlus* should be the last menu item in the *Tools* menu.

## Modifying the Sheet Size

The following directives in the `translate.cpm` file define the sheet sizes:

```
CONCEPT_SYMBOL_SHEET_A '-3875,5125,125,-125'  
CONCEPT_SYMBOL_SHEET_B '-8125,5125,125,-125'  
CONCEPT_SYMBOL_SHEET_C '-10750,8275,0,0'  
CONCEPT_SYMBOL_SHEET_D '-14875,11000,2100,0'  
CONCEPT_SYMBOL_SHEET_E '-10950,8250,11000,-8700'  
CONCEPT_SYMBOL_SHEET_F '-9975,6975,10000,-6975'
```

New sheet sizes can be added by following the syntax given below:

```
CONCEPT_SYMBOL_SHEET_<Sheet_Size_Name> '<Upper Left X,Y Co-ord> <Lower Right X,Y  
Co-ord>
```

**Note:** The `translate.cpm` file is located in the `<CDS_SITE>\share\cdssetup\LMAN` location.

To make the new sheet size available from the *Tools – Setup* option, you need to update the `ItemList` entries in the `SymbolSetup.panel` file located in the `<CDS_SITE>\share\cdssetup\LMAN` location with the new values.

For example, you want to add a new sheet size called `my_sheet` to Part Developer. To do so:

1. Add the following line in the `translate.cpm` file:

```
CONCEPT_SYMBOL_SHEET_my_sheet '-10000,10000,10000,-10000'
```

2. Next, update the `ItemList` entry in the `SymbolSetup.panel` file with the `my_sheet` entry. The `ItemList` should now read as:

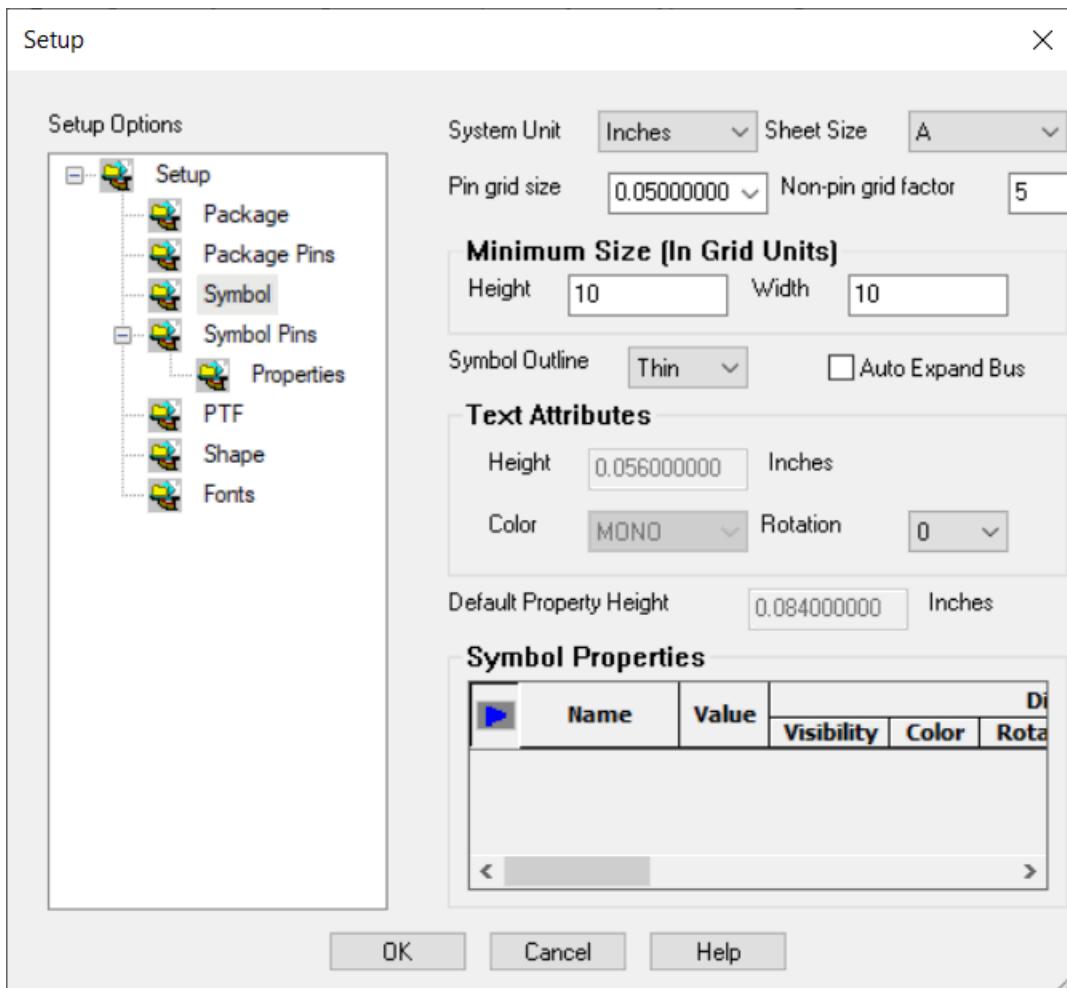
```
(ItemList A,B,C,D,E,F,my_sheet,MAX_SIZE)
```

3. Save the files and start Part Developer.

## Part Developer User Guide

### Advanced Tasks

4. Open a project and select *Tools – Setup*.
5. Click *Symbol* in the *Setup Options* tree.
6. Click on the *Sheet Size* drop-down menu. The new sheet size should be visible in the drop-down list.



## Modifying RefDes Prefix and Class Values

The values for RefDes Prefix and CLASS can be modified, added, or deleted by making changes to the following entries in the `propfile.prop` file:

```
(PackageRefDes  
  (PreDefined "U,C,R,I,J,T,X,D,M")
```

```
)  
(PackageClass  
  (PreDefined "IC, IO, DISCRETE, MECHANICAL")  
)
```

**Note:** The `propfile.prop` file is located in the `<CDS_SITE>\share\cdssetup\LMAN` location.

After making the changes in the file, save the file and restart Part Developer. The new or modified values will now be available.

## Modifying the Default List of Package, Package Pin, Symbol, and Symbol Pin Properties

The default list of package, package pin, symbol, and symbol pin properties can be modified, added, or deleted by making changes to the following entries in the `propfile.prop` file:

```
(Package  
  (PreDefined  
    "FAMILY, DEFAULT_SIGNAL_MODEL, TECH, POWER_GROUP, REF_DES_PATTERN, DESCRIPTION, BODY_NAME")  
    (NotAllowed  
      "ALT_SYMBOLS, CLASS, JEDEC_TYPE, NC_PINS, PART_NAME, PHYS_DES_PREFIX, POWER_PINS")  
    (Filtered  
      "ALT_SYMBOLS, CLASS, JEDEC_TYPE, NC_PINS, PART_NAME, PHYS_DES_PREFIX, POWER_PINS")  
  )  
(PackagePin  
  (PreDefined "PIN_GROUP")  
  (NotAllowed  
    "ALLOW_CONNECT, BIDIRECTIONAL, INPUT_LOAD, OUTPUT_LOAD, NO_ASSERT_CHECK,  
    NO_DIR_CHECK, NO_IO_CHECK, NO_LOAD_CHECK, OUTPUT_TYPE, PIN_TYPE, PINUSE,  
    UNKNOWN_LOADING, PIN_NUMBER")  
  (Filtered  
    "ALLOW_CONNECT, BIDIRECTIONAL, INPUT_LOAD, OUTPUT_LOAD, NO_ASSERT_CHECK,  
    NO_DIR_CHECK, NO_IO_CHECK, NO_LOAD_CHECK, OUTPUT_TYPE, PIN_TYPE, PINUSE,  
    UNKNOWN_LOADING, PIN_NUMBER")
```

## Part Developer User Guide

### Advanced Tasks

---

```
(UI_Predefined "")  
)  
  
(Symbol  
  
(PreDefined  
"PACK_TYPE,VALID_PACK_TYPE,JEDEC_TYPE,ALT_SYMBOLS,PART_NAME,$LOCATION,SPLIT_  
INST,SPLIT_INST_NAME")  
  
(NotAllowed "HAS_FIXED_SIZE")  
  
(Filtered "HAS_FIXED_SIZE")  
  
(Duplicate "BUBBLE_GROUP","POWER_GROUP")  
)  
  
(SymbolPin  
  
(PreDefined "$PN,PIN_NUMBER")  
  
(NotAllowed "PIN_TEXT")  
  
(Filtered "PIN_TEXT")  
)
```

There are four keywords that need to be understood to ensure that the changes are correct.

■ PreDefined

These are the default list of properties available with Part Developer. All new properties need to be appended to this list.

■ NotAllowed

The properties mentioned in this list cannot be added as a property. Part Developer will generate errors if these properties are added. You can append to this list to customize your list of invalid properties.

■ Filtered

Same as NotAllowed

■ Duplicate

Properties in this list can be added multiple times.

■ UI\_Predefined

This keyword is valid only for Package Editor. This keyword enables you to add a property column. To add multiple columns, use a comma-separated list. For example, if you want PIN\_GROUP and DIFFERENTIAL\_PAIR property columns to be always visible, add the following:

UI\_Predefined "PIN\_GROUP, DIFFERENTIAL\_PAIR"

**Note:** The propfile.prop file is located in the `<CDS_SITE>\share\cdssetup\LMAN` location.

After making the changes in the file, save the file and restart Part Developer. The new or modified values will now be available.

## Configuring Translation Rules for Import and Export

Part information is present in different formats and syntaxes in different data sources, such as XML files, CSV files and so on. Some of these formats, naming conventions, or characters may not be supported in the Cadence flow. For example, pin loading values are mandatory in Design Entry HDL but not in Capture, and by default, they are put as user properties in Capture. Part Developer translates such information sets when importing data.

The default translation rules are provided in a file called `translate.cpm`, located in `<CDS_SITE>\share\cdssetup\LMAN`. You can modify this file to add your own translation rules.

The `translate.cpm` file starts with the keyword `START_LMAN_CONFIG` and ends with the keyword `END_LMAN_CONFIG`. The general syntax followed for defining rules in this file is the following:

```
<Domain Name>_<ObjectType>_<Action ID> <Action Name> <Action Attributes>
  <Condition Name> <Condition Attribute> <Parameters>
```

---

Argument	Description
Domain Name	<ul style="list-style-type: none"><li>■ CAPTURE when import data source is in Capture format</li><li>■ CONCEPT when import data source is in Design Entry HDL format</li><li>■ XMLIN when input data source is in ETools XML format</li><li>■ XMLOUT when output data source is in ETools XML format</li></ul>
ObjectType	Name of the object on which the action needs to be applied. For example, PROPERTY, SYMPORT, PHYSPORT, and so on.  SYMPORT stands for ports on a symbol.  PHYSPORT stands for ports on a package.

## Part Developer User Guide

### Advanced Tasks

---

Argument	Description
Action ID	A unique number with the ACTION_ prefix to define the rule.
Action Name	Action that needs to be performed. For example, STR_REPLACE, STR_INSERT, STR_REMOVE, STR_APPEND, DELETEPROP, and STR_REPLACELAST.
Action Attribute	Type of attribute for <i>Action Name</i> . For example, NAME, PROPVALUE, BASENAME, and PROPVALUE.
Condition Name	Type of the condition that needs to be checked while applying the action. For example, STR_COMPARE and STR_COMPARENOCASE.
Condition Attribute	Type of attribute of “Condition Name”. For example, ‘NAME’, ‘’. Space represents anything i.e., *.
Parameters	Maximum of three parameters are possible to be followed by Condition Attributes. The number of parameters that needs to be specified depends on the value of the Action Name argument being applied.

---

### Example 1

Consider the following entry in the translate.cpm file:

```
CAPTURE_PROPERTY_ACTION_1 'STR_REPLACE' 'NAME' 'STR_COMPARE' 'NAME'  
    'Implementation Path' 'IMPLEMENTATION_PATH' 'Implementation Path'.
```

Let us now correlate the above statement with the general syntax:

```
<Domain Name>_<ObjectType>_<Action ID> <Action Name> <Action Attributes>  
    <Condition Name> <Condition Attribute> <Parameters>
```

:

Argument	Description
Domain Name	CAPTURE
ObjectType	PROPERTY

## Part Developer User Guide

### Advanced Tasks

---

Argument	Description
Action ID	ACTION_1  The unique identifier assigned to this action.
Action Name	STR_REPLACE  Indicates that the action to be done is a string-replace.
Action Attribute	NAME  Indicates that the attribute of this action is NAME.
Condition Name	'STR_COMPARE'  Indicates that the condition to take action is to compare a string.
Condition Attribute	NAME  Indicates that the attribute of this condition is NAME
Parameters	'Implementation Path' 'IMPLEMENTATION_PATH' 'Implementation Path'  It has three parameters. This indicates that the first parameter is an action attribute value, the third parameter is a string comparison attribute value, and the second parameter is a replace with attribute value. So when a property 'Implementation Path' is found, it is replaced with a property called 'IMPLEMENTATION_PATH'.

---

### Example 2

Translate.cpm entry:

```
CAPTURE_PROPERTY_ACTION_7 'STR_INSERT' 'PROPVVALUE' 'STR_COMPARENOCASE' 'NAME' '0'
  '(' 'NC_PINS'
```

#### Meaning:

Insert "(" at 0 th location of NC\_PINS property. Also, do case-insensitive comparison when checking for the property.

### Example 3

Translate.cpm entry:

## Part Developer User Guide

### Advanced Tasks

---

```
CAPTURE_PROPERTY_ACTION_8 'STR_APPEND' 'PROPVVALUE' 'STR_COMPARENOCASE' 'NAME' '')'  
'NC_PINS'
```

#### **Meaning:**

Append ")" to the NC\_PINS property. Also, do case insensitive comparison when checking for the property.

#### **Example 4**

Translate.cpm entry:

```
CAPTURE_SYMPORT_ACTION_1 'DELETEPROP' '' 'STR_COMPARENOCASE' 'NAME' 'Name'
```

#### **Meaning:**

On a symbol port, delete the "Name" property if it exists in the input file.

#### **Example 5**

Translate.cpm entry:

```
CAPTURE_SYMPORT_ACTION_9 'STR_REPLACE' 'BASENAME' '' '' '>' '_GT_'
```

#### **Meaning:**

Replace '>' with '\_GT\_' on symbol port.

**Note:** When you change it on symbol port, ensure that the physical port is also updated to maintain the packageability of the symbol.

#### **Example 6**

Translate.cpm entry:

```
CAPTURE_PHYSPORT_ACTION_1 'DELETEPROP' '' 'STR_COMPARENOCASE' 'NAME' 'Name'
```

#### **Meaning:**

On a physical port, delete the "Name" property if it exists in the input file.

#### **Example 7**

Translate.cpm entry:

## Part Developer User Guide

### Advanced Tasks

---

```
CAPTURE_PHYSPORT_ACTION_2 'STR_REPLACE' 'BASENAME' '' '' '>' '_GT_'
```

#### **Meaning:**

Replace '>' with '\_GT\_' on package port.

**Note:** You will need to update the corresponding symbol's physical port too.

#### **Example 8**

Translate.cpm entry:

```
CAPTURE_PACKAGE_ACTION_1 'DELETEPROP' '' 'STR_COMPARE' 'NAME' 'Name'
```

#### **Meaning:**

Delete the "Name" package property.

#### **Example 9**

Translate.cpm entry:

```
CAPTURE_ALIAS_ACTION_1 'STR_REPLACELAST' 'NAME' '' '' '/' '_'
```

#### **Meaning:**

Replace the last '/' in alias names with '\_'.

## **Configuring Mentor Export**

### **Adding New Properties**

You can add new properties along with their values by using the following directive:

```
MENTOROUT_SYMBOL_ACTION_1 'ADDPROP' '<property_name>:<property_value>'
```

### **Initializing Properties**

The following directive initializes a property with a specified string:

```
MENTOROUT_SYMBOL_ACTION_1 'FORMATPROPVAL' '<property_name>:<string>'
```

**Example**

The following directive initializes the \$LOCATION property with ?:

```
MENTOROUT_SYMBOL_ACTION_1 'FORMATPROPVAL' '$LOCATION:?'
```

**Specifying the Default Offset for Moving Properties**

The following directive specifies the default offset for moving properties:

```
MENTOROUT_PROPMOVEOFFSET_"x:y"
```

**Example**

The following directive specifies 400, 400 as the default offset for moving properties:

```
MENTOROUT_PROPMOVEOFFSET_"400:400"
```

**Note:** The x and y values are specified in Design Entry HDL grid units.

**Moving a Property with Respect to Another Property**

The following directive moves *property1* by *x* and *y* grid units with respect to *property2* in a symbol:

```
MENTOROUT_SYMBOL_ACTION_7 'MOVEPROPLOC_WRT_REFPROPLOC'  
'<property1>:<property2>:<x>:<y>'
```

**Note:** If you do not specify the values for x and y, Part Developer uses the x and y values specified using the PROPMOVEOFFSET directive.

**Copying Properties from the Package to the Symbol**

The following directive copies properties from the package to the symbol:

```
MENTOROUT_COPYPROPPKGTOSYM_"<property_name>"
```

**Assigning a Property Value by Concatenating the Values of Specified Properties**

The following directive concatenates the values of specified properties and assigns the concatenated value to a specified property:

```
MENTOROUT_SYMBOL_ACTION_1 'APPENDPROPVAL'  
'<property_name>:<propvalue_1>:<propvalue_2>'
```

### **Example**

The following directive assigns U5 to property *REF* if properties *CLASS* and *INSTANCE* have values U and 5, respectively:

```
MENTOROUT_SYMBOL_ACTION_1 'APPENDPROPVAL' 'REF:CLASS:INSTANCE'
```

**Note:** The properties that are to be concatenated must exist in `symbol.css` or must be added using the `ADDPROP` action in `translate.cpm`.

### **Deleting Existing Properties**

The following directive deletes a specified property:

```
MENTOROUT_SYMBOL_ACTION_9 'DELETEPROP' '' 'STR_COMPARE' 'NAME' '<  
property_to_be_deleted>'
```

### **Unassociating Pin Text**

The following directive in `translate.cpm` unassociates pin text and enables pin text color to be set according to the default Mentor settings:

```
MENTOROUT_SYMPORT_ACTION_2 'UNASSOCIATE_PIN_TEXT'
```

## **Configuring the Predefined Headers for CSV Import**

By default, the following package and pin properties are predefined as headers for CSV import:

- `package_name`
- `jedec_type`
- `assertion_char`
- `load_setupfile`
- `pin_name`
- `pin_number`
- `pin_type`
- `assertion`
- `pin_location`

- pin\_position
- symbol
- pin\_shape
- diff\_pair\_pins\_pos
- diff\_pair\_pins\_neg

The definitions are controlled through the following directives in the `cds.cpm` file.

```
Import_Csv_Replace_packagename 'package_name'  
Import_Csv_Replace_assertionchar 'assertion_char'  
Import_Csv_Replace_jedectype 'jedec_type'  
Import_Csv_Replace_loadsetupfile 'load_setupfile'  
Import_Csv_Replace_pinname 'pin_name'  
Import_Csv_Replace_pinnumber 'pin_number'  
Import_Csv_Replace_pintype 'pin_type'  
Import_Csv_Replace_assertion 'assertion'  
Import_Csv_Replace_pinlocation 'pin_location'  
Import_Csv_Replace_pinposition 'pin_position'  
Import_Csv_Replace_symbol 'symbol'  
Import_Csv_Replace_pinshape 'pin_shape'  
Import_Csv_Replace_DIFFPAIRPINSPOS 'DIFF_PAIR_PINS_POS'  
Import_Csv_Replace_DIFFPAIRPINSNEG 'DIFF_PAIR_PINS_NEG'
```

**Note:** The `cds.cpm` file is located in `<CDS_SITE>/share/cdssetup/projmgr`.

You can modify the directives to change your own headers for CSV import. For example, if you want to ensure that data under the column `pkg_name` should appear as the package name, you need to update the `Import_Csv_Replace_packagename` directive as follows:

```
Import_Csv_Replace_packagename 'pkg_name'
```



***You can modify only the header names that represent the entries in the CSV file. You cannot add your own headers by adding new directives.***

## Configuring the Predefined Headers for Text Import

By default, the following headers are pre-defined for text import:

- pin\_name
- pin\_number
- pin\_type
- pin\_location
- pin\_position
- symbol

The definitions of these headers are mentioned in the `propfile.prop` file under `ImportText`.

```
(ColName
  (pin_name "Pin Name,Pin_name")
  (pin_number "Pin no,Pin number,pin_no,PN,pin_number")
  (pin_type "Pin type,direction,pin_type")
  (pin_location "Pin Location,location,pin_location,pin_loc")
  (pin_position "Pin Position,position,pin_position,pin_pos")
)
```

**Note:** The `propfile.prop` file is located at `<CDS_SITE>/share/cdssetup/lman`.

You can modify the header definitions according to your requirements or preferences. For example, if you want to ensure that the data for the `pin-name` column in the text file is imported for the `pin_name` header, you need to add the following definition in the `propfile.prop`:

```
(ColName
  (pin_name "Pin Name,Pin_name,pin-name")
)
```

## Configuring Pin Text Position Defaults

The pin text position defaults can be configured by changing the values of the following directives in the `cds.cpm` file

```
Symbol_Pintext_TopBottom_XOffset '0'  
Symbol_Pintext_TopBottom_YOffset '10'  
Symbol_Pintext_LeftRight_XOffset '10'  
Symbol_Pintext_LeftRight_YOffset '0'
```

The following two directives control the default placement of text for left and right pins.

```
Symbol_Pintext_LeftRight_XOffset '10'  
Symbol_Pintext_LeftRight_YOffset '0'
```

For example, changing above values as below, moves pin text 10 units away on the x axis and 10 units below than the original on the y axis.

```
Symbol_Pintext_LeftRight_XOffset '20'  
Symbol_Pintext_LeftRight_YOffset '-10'
```

The same is true for top and bottom pins with following directives.

```
Symbol_Pintext_TopBottom_XOffset '0'  
Symbol_Pintext_TopBottom_YOffset '10'
```

## Configuring to Use Square Brackets in Vector Pin Names

By default, Part Developer displays vector pin bits in angular brackets (`<>`). If you want to display vector pin bits in square brackets (`[ ]`) instead of angular brackets (`<>`), you can specify the following directive in the global section of the `cds.cpm` file at the `CDS_SITE` location:

```
VectorSqrBracket '1'
```

However, in `chips.prt`, vector pin names are written using angular brackets (`<>`) regardless of the `VectorSqrBracket` setting.

## Flow Impact of Pins with Square Brackets in Basenames

If `VectorSqrBracket` is set to 1, Part Developer does not support the use of square brackets in the basenames of pins. This behavior is same as the behavior of the Allegro schematic designer Design Entry HDL when the `MULTI_FORMAT` directive in `cds.cpm` is set to ON. For

example, if the basename of a vector pin is U+ [O] , the pin name is considered invalid in both Part Developer and Design Entry HDL if `VectorSqrBracket` is set to 1 and `MULTI_FORMAT` is set to ON.

## Importing Pins with Square Brackets in Basenames

If the `VectorSqrBracket` directive is set to 1, to import CSV or text files that contain pins with square brackets in their basenames, you need to set the `Import_Csv_ApplyVectorConversion` directive to TRUE in the `cds.cpm` file.

Alternatively, you can convert pins with square brackets ([]) to scalar pins by adding port translation rules in `translate.cpm`.

```
CSV_SYMPORT_ACTION_11 'STR_REPLACE' 'BASENAME' '' '' '[' '_RB_'
CSV_SYMPORT_ACTION_12 'STR_REPLACE' 'BASENAME' '' '' ']' '_LB_'
CSV_PHYSPORT_ACTION_11 'STR_REPLACE' 'BASENAME' '' '' '[' '_RB_'
CSV_PHYSPORT_ACTION_12 'STR_REPLACE' 'BASENAME' '' '' ']' '_LB_'
```

**Note:** The alternative method is applicable in all types of import.

## Configuring the Alignment of Symbol Properties

You can configure the alignment of symbol properties by changing the values of the following directive in `setup.cpm`:

`Symbol_SymProperty_Setup_Apply`

If the value of the `Symbol_SymProperty_Setup_Apply` directive is set to `Yes`, symbol property alignment is determined from the setup.

If the value of the `Symbol_SymProperty_Setup_Apply` directive is set to `No`, symbol property alignment is determined in the following way:

---

<b>Location</b>	<b>Alignment</b>
Top-Left	Right
Left	
Bottom-Left	

Top	Center
Center	
Bottom	
Top-Right	Left
Right	
Bottom-Right	

## Configuring the Alignment of Symbol Pin Properties

You can configure the alignment of properties for symbol pins on the left, right, top, and bottom of the symbol outline by changing the values of the following directives in `setup.cpm`:

```
Symbol_PinProperty_Alignment_Left 'Right'  
Symbol_PinProperty_Alignment_Right 'Left'  
Symbol_PinProperty_Alignment_Top 'Left'  
Symbol_PinProperty_Alignment_Bottom 'Right'
```

**Note:** The above directives work only when the `Symbol_PinPropertySetup_Alignment_Apply` directive in `setup.cpm` is set to `No`. To define the alignment settings at a project level through *Tools – Setup – Symbol Pins – Properties*, make sure that the `Symbol_PinPropertySetup_Alignment_Apply` directive in `setup.cpm` is set to `Yes`.

## Configuring the Rotation of Symbol Pin Properties

You can configure the rotation of properties for symbol pins on the left, right, top, and bottom of the symbol outline by changing the values of the following directives in `setup.cpm`:

```
Symbol_PinProperty_Rotation_Left '0'  
Symbol_PinProperty_Rotation_Right '0'  
Symbol_PinProperty_Rotation_Top '90'  
Symbol_PinProperty_Rotation_Bottom '90'
```

**Note:** The above directives work only when the `Symbol_PinPropertySetup_Rotation_Apply` directive in `setup.cpm` is set to `No`. To define the rotation settings at a project level through *Tools – Setup – Symbol Pins – Properties*, make sure that the `Symbol_PinPropertySetup_Rotation_Apply` directive in `setup.cpm` is set to `Yes`.

## Configuring the Placement and Stackup of Symbol Properties

You can configure the position of the first symbol property with each type of location by changing the values of the following CPM directives:

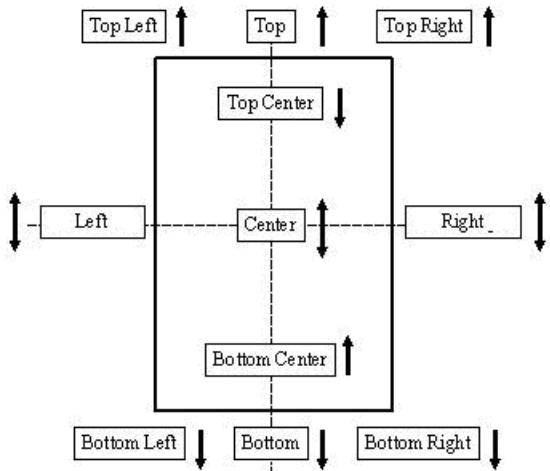
Directive	Description
Symbol_Property_Outline_OffsetTop '50'	Specifies the number of Symbol Editor grid units used to determine the offset of the first symbol property with the <i>Location</i> value <i>Top</i> with respect to the symbol outline  The property is placed on the y axis.
Symbol_Property_Outline_OffsetLeft '50'	Specifies the number of Symbol Editor grid units used to determine the offset of the first symbol property with the <i>Location</i> value <i>Left</i> with respect to the symbol outline  The property is placed on the x axis.
Symbol_Property_Outline_OffsetRight '50'	Specifies the number of Symbol Editor grid units used to determine the offset of the first symbol property with the <i>Location</i> value <i>Right</i> with respect to the symbol outline  The property is placed on the x axis.
Symbol_Property_Outline_OffsetBottom '50'	Specifies the number of Symbol Editor grid units used to determine the offset of the first symbol property with the <i>Location</i> value <i>Bottom</i> with respect to the symbol outline  The property is placed on the y axis.

**Note:** The position of the first symbol property with the *Location* value *Top-Left*, *Top-Right*, *Bottom-Left*, or *Bottom-Right* is derived using the respective pair of directives. For example, the position — x, y coordinates — of the first symbol property with the *Location* value *Top-Left* is derived using the directives `Symbol_Property_Outline_OffsetTop` and `Symbol_Property_Outline_OffsetLeft`.

## Part Developer User Guide

### Advanced Tasks

If more than one symbol property has the same location, Part Developer stacks the properties in the following way:



You can control the space between two adjacent properties by configuring the following directive:

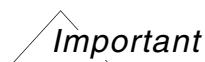
`Symbol_Property_Stacking_Offset '15'`

---

## **Part Developer Console Commands**

---

Part Developer provides the ability to run in a non-GUI mode through a set of console commands, which you specify at the command prompt. This chapter describes the commands and their functions.



The console commands are available only with the highest license tier.

### **csv2ptf**

Creates a part table file from a CSV file

#### **csv2ptf Process Overview**

To create a part table file from a CSV file:

1. Specify setup.cpm in the name of the field that should be used to create part names in the PTF file. The directive to be used is Import\_Csv\_PtfLogPart.
2. Configure default configuration rules if required.  
For more information on csv2ptf configuration rules in setup.cpm, see [Configuring Conversion Rules for csv2ptf Import](#) on page 366
3. Sort the source CSV file on the column that should be used to create part names in the PTF file.  
Note: This is a recommended step, which improves the readability and findability of data in the PTF file.
4. Create the configuration file to define key and injected properties.



Part Developer reads the field names from the configuration file and not the source CSV file.

For information on how to create the configuration file for a source CSV file, see [Creating a Configuration File for csv2ptf Import](#) on page 366.

**5. Specify the csv2ptf command.**

For information on the syntax, see [Usage](#) on page 367.

## Configuring Conversion Rules for csv2ptf Import

You can control the behavior of `csv2import` by configuring the conversion rules defined in `setup.cpm`. The following table describes the directives that are used to define `csv2ptf` conversion rules:

Directive	Function	Example
<code>Import_Csv_PtfLogPart</code>	Specifies the name of the column in the source CSV file that should be used to create part names in the PTF file	<code>Import_Csv_PtfLogPart 'Part_Number'</code>
<code>Import_Csv_PtfSeparationChar</code>	Specifies the characters that will be used to enclose field names in the PTF file	<code>Import_Csv_PtfSeparationChar ''</code>
<code>Import_Csv_PtfDelimiterChar</code>	Specifies the character that will be used to separate field names in the PTF file	<code>Import_Csv_PtfDelimiterChar '/'</code>

## Creating a Configuration File for csv2ptf Import

To create the configuration file that can be used to convert a CSV file to a PTF file:

**1. Copy the source CSV file and rename the copied file.**

## Part Developer User Guide

### Part Developer Console Commands

2. Delete the data rows.
3. For each column header, specify KEY or INJECTED in the cell below the column header.

	A	B	C	D	E	F	G	H	I	J
1	VALUE	TOLERAN	VOLTAGE	MATERIAL	PACK_TYPE	PART_NU	JEDEC_TYP	ALT_SYM	CLASS	DESCRIPTION
2	KEY	KEY	KEY	KEY	KEY	INJECTED	INJECTED	INJECTED	INJECTED	INJECTED
3										
4										

## Usage

```
csv2ptf
  -csvfile <source_csv_file_path>/<csv_file_name>.csv
  -configfile <source_config_file_path>/<config_file_name>.csv
  -ptffile <target_ptf_file_path>/<ptf_file_name>.ptf
  -product pcb_librarian_expert
```

## Parameter Description

---

Parameter	Description
csvfile	Absolute path to the source CSV file

---

## Part Developer User Guide

### Part Developer Console Commands

---

Parameter	Description
configfile	Absolute path to the source configuration file  The configuration file categorizes the fields in the source CSV file into key and injected properties.  For information on how to create the configuration file for a source CSV file, see <a href="#">Creating a Configuration File for csv2ptf Import</a> on page 366.
ptffile	Absolute path to the target PTF file

## Example

The following example creates a PTF file, *LIB1.ptf*, from the CSV file *LIB1.csv* based on the key and injected property definition provided in the configuration file *LIB1\_cfg.csv*.

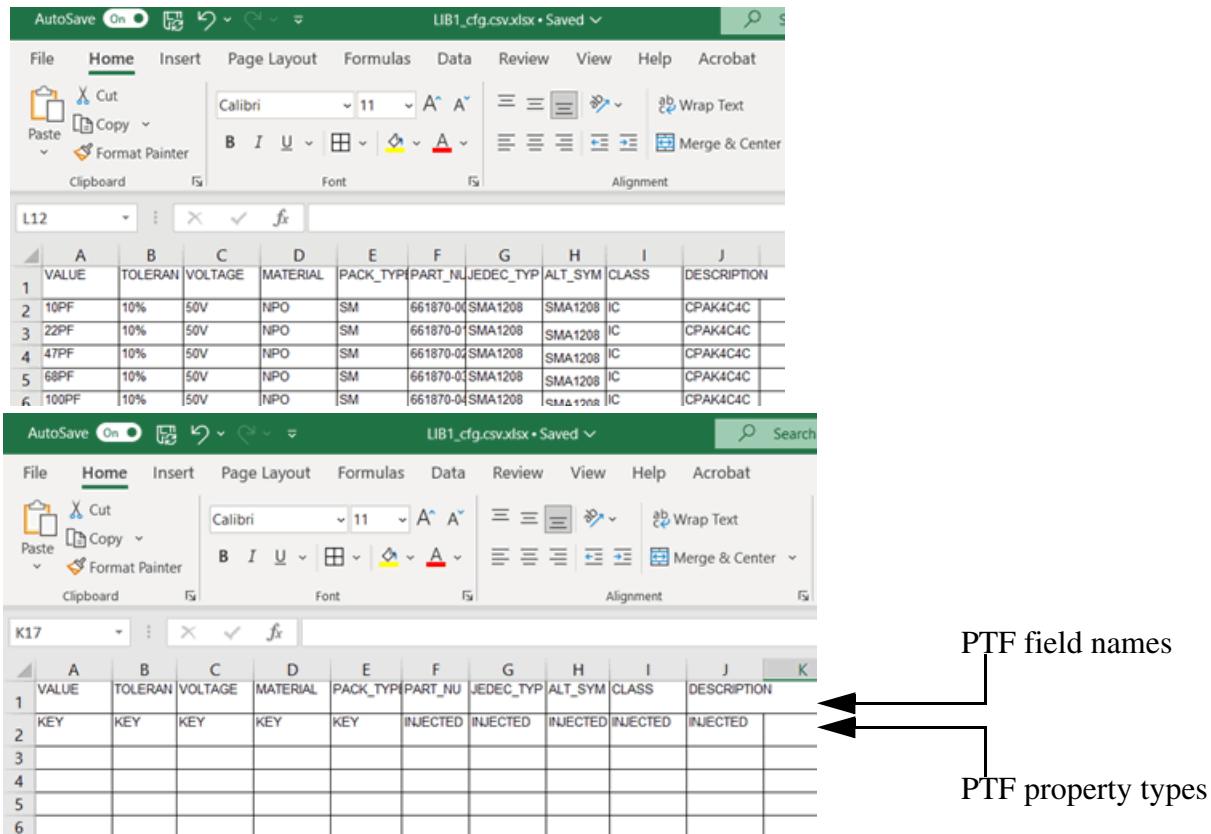
```
csv2ptf.exe -product pcb_librarian_expert -csvfile D:\csv2ptf\csvs\LIB1.csv -  
ptffile D:\csv2ptf\ptfs\LIB1.ptf -configfile D:\csv2ptf\cfgs\LIB1_cfg.csv
```

## Part Developer User Guide

### Part Developer Console Commands

---

The following graphic displays a snapshot of the source CSV file and the configuration file:



**Source CSV File (Top Sheet):**

	A	B	C	D	E	F	G	H	I	J
1	VALUE	TOLERAN	VOLTAGE	MATERIAL	PACK_TYP	PART_NU	JEDEC_TYP	ALT_SYM	CLASS	DESCRIPTION
2	10PF	10%	50V	NPO	SM	661870-0	SMA1208	SMA1208	IC	CPAK4C4C
3	22PF	10%	50V	NPO	SM	661870-01	SMA1208	SMA1208	IC	CPAK4C4C
4	47PF	10%	50V	NPO	SM	661870-02	SMA1208	SMA1208	IC	CPAK4C4C
5	68PF	10%	50V	NPO	SM	661870-03	SMA1208	SMA1208	IC	CPAK4C4C
6	100PF	10%	50V	NPO	SM	661870-04	SMA1208	SMA1208	IC	CPAK4C4C

**Configuration File (Bottom Sheet):**

	A	B	C	D	E	F	G	H	I	J	K
1	VALUE	TOLERAN	VOLTAGE	MATERIAL	PACK_TYP	PART_NU	JEDEC_TYP	ALT_SYM	CLASS	DESCRIPTION	
2	KEY	KEY	KEY	KEY	KEY	INJECTED	INJECTED	INJECTED	INJECTED	INJECTED	
3											
4											
5											
6											

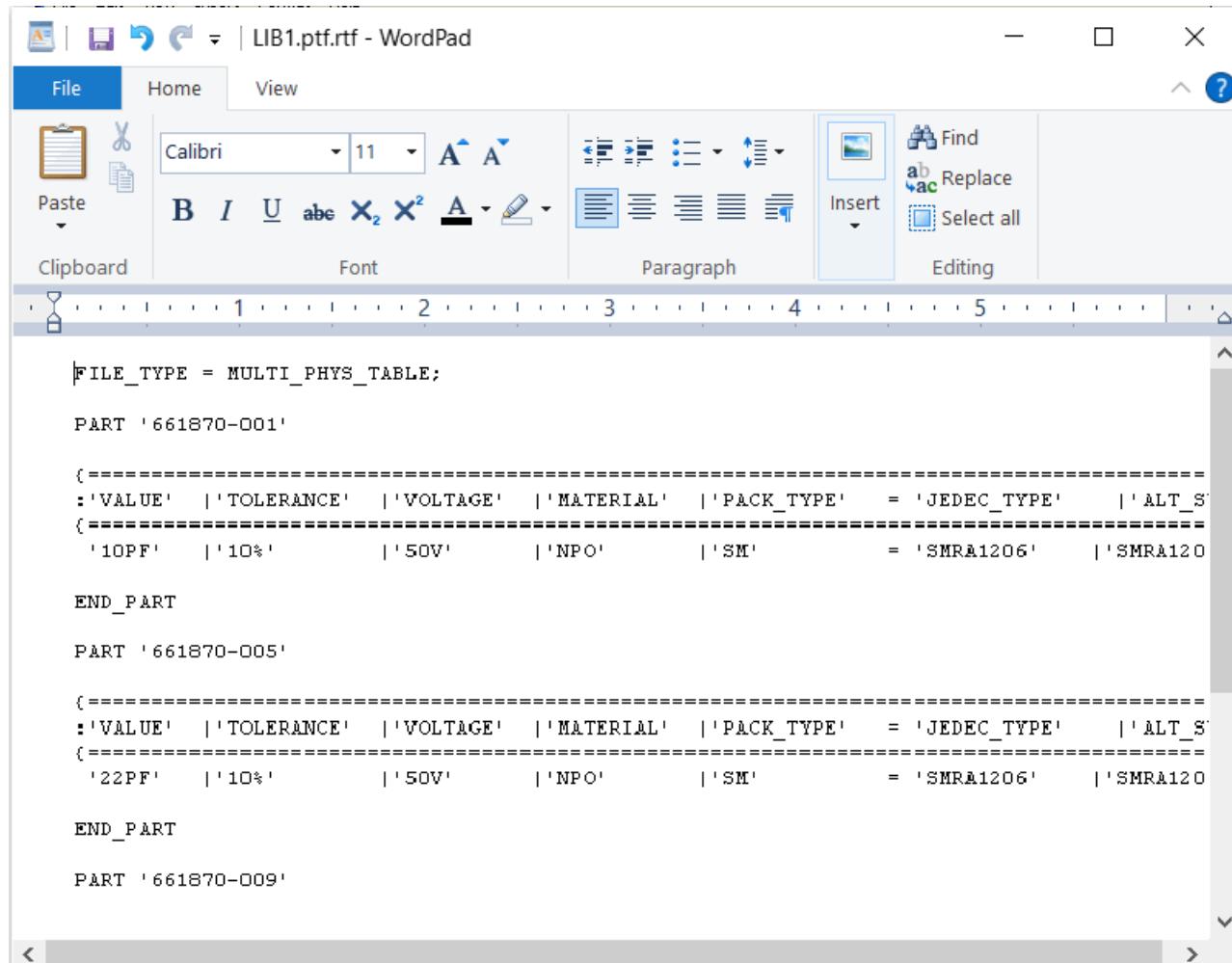
**Annotations:**

- An arrow points from the **K** column to the heading **PTF field names**.
- An arrow points from the **K** column to the heading **PTF property types**.

## Part Developer User Guide

### Part Developer Console Commands

The following graphic displays a snapshot of the PTF file created using the configuration file and source CSV file shown in this example:



A screenshot of the Microsoft WordPad application window titled "LIB1.ptf.rtf - WordPad". The window shows the RTF content of a PTF file. The content includes several PART definitions for different components, each with specific parameters like TOLERANCE, VOLTAGE, MATERIAL, and PACK\_TYPE. The WordPad interface is visible at the top, showing the ribbon tabs (File, Home, View) and various toolbar icons for font, paragraph, and editing.

```
FILE_TYPE = MULTI_PHYS_TABLE;

PART '661870-001'
{
  =====
  : 'VALUE'  || 'TOLERANCE'  || 'VOLTAGE'  || 'MATERIAL'  || 'PACK_TYPE'  = 'JEDEC_TYPE'  || 'ALT_S'
  =====
  '10PF'    || '10%'       || '50V'      || 'NPO'       || 'SM'        = 'SMRA1206'  || 'SMRA120
}
END_PART

PART '661870-005'
{
  =====
  : 'VALUE'  || 'TOLERANCE'  || 'VOLTAGE'  || 'MATERIAL'  || 'PACK_TYPE'  = 'JEDEC_TYPE'  || 'ALT_S'
  =====
  '22PF'    || '10%'       || '50V'      || 'NPO'       || 'SM'        = 'SMRA1206'  || 'SMRA120
}
END_PART

PART '661870-009'
```

## con2con

Creates a Design Entry HDL part from another Design Entry HDL part.

### Usage

#### ***Non-ECO Mode***

con2con

## Part Developer User Guide

### Part Developer Console Commands

---

```
-product pcb_librarian_expert
-proj <path_for_proj_Lib>/<proj_file_name>.cpm
-cdslib <path_for_cds.lib>/cds.lib
-lib <source_design_entry_HDL_lib_name>
-cell <source_design_entry_HDL_cell_name>
[-outlib <destination_design_entry_HDL_lib_name>]
[-outcell <destination_design_entry_HDL_cell_name>]
[-overwrite]
[-metadataonly]
[-baselineonly]
[-baseline_clean]
[-verifonly]
[-ignoreFootprint]
[-ignorePTF]
```

#### ***ECO Mode***

con2con

```
-product pcb_librarian_expert
-proj <path_for_proj_Lib>/<proj_file_name>.cpm
-cdslib <path_for_cds.lib>/cds.lib
-lib <source_design_entry_HDL_lib_name>
-cell <source_design_entry_HDL_cell_name>
-ecolib <name_of_eco_Lib>
-ecocell <name_of_eco_Cell>
[-IgnorePropDeletion]
[-autocreatediffpair]
```

#### **Parameter Description**

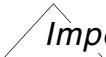
---

<b>Parameter</b>	<b>Description</b>
proj	Absolute path to the project file.
cdslib	Absolute path to the <code>cds.lib</code> file.
lib	Source Design Entry HDL library name.
cell	Source Design Entry HDL cell name.
outlib	Destination Design Entry HDL library name.

## Part Developer User Guide

### Part Developer Console Commands

---

<b>Parameter</b>	<b>Description</b>
outcell	Destination Design Entry HDL cell name.
metadataonly	<Optional> Generates metadata view only. This can be used to generate the metadata view for the parts created on 14.x release or below.
baselineonly	<Optional> Baselines the cell.
baseline_clean	<Optional> Clears all the existing revision data from metadata view except the <code>pdv_validation.txt</code> file.
verifyonly	<Optional> Runs all the Part Developer verification checks that happen on save. A log file, <code>pdv_validation.txt</code> , is generated and stored in the metadata view.
overwrite	<Optional> Overwrites an existing cell.
<p> <i>Important</i></p> <p>An error message is displayed if this parameter is not specified and there already exists a cell with the same name in the destination library. The destination cell is not overwritten by default. It is strongly suggested that you use the outlib and outcell parameters when using the overwrite parameter. Not using the outlib and outcell parameters may result in the deletion of the views of the current cell.</p>	
ecolib	Destination ECO library name.
ecocell	Destination ECO cell name.
IgnorePropDeletion	Retains the properties that exist in the destination cell but are not available in the source ECO file.
ignoreFootprint	Ignores the JEDEC_TYPE values specified for the primitives.
ignorePTF	Ignores part table rows.
autocreatediffpair	Adds differential pair properties to pins based on naming rules specified through the <code>DiffPair_Recognition_Rules</code> directive in <code>cds.cpm</code> .
	For more information on the <code>DiffPair_Recognition_Rules</code> directive, see <a href="#">Specifying a Naming Convention for Autocreation of Differential Pairs</a> on page 130.

---

## Example

The following example creates a copy of the Design Entry HDL part *s5920* in the *ulab\_proj\_lib* library. The new part created in the same directory is called *ss59*.

```
con2con -proj d:/ulab/ulab_proj.cpm -cdslib d:/ulab/cds.lib -lib ulab_proj_lib -  
cell s5920 -outlib ulab_proj_lib -outcell ss59 -product pcb_librarian_expert
```

## viewlogic2con

Creates a Design Entry HDL part from a ViewLogic part.

## Usage

### **Non-ECO Mode**

```
viewlogic2con  
-proj <path_for_proj_Lib>/<proj_file_name>.cpm  
-cdslib <path_for_cds.lib>/cds.lib  
-viewlogicdir <directory with ViewLogic files>  
-package <ViewLogic file name without extension>  
-lib <design_entry_HDL_library_name>  
[-cell <design_entry_HDL_cell_name>]  
[-overwrite]  
-product pcb_librarian_expert
```

**Note:** For library translation, use the following syntax:

```
viewlogic2con  
-product "PCB_LIBRARIAN_EXPERT"  
-proj <path_for_proj_Lib>/<proj_file_name>.cpm  
-cdslib <path_for_cds.lib>/cds.lib  
-viewlogicdir <directory with ViewLogic files>  
-lib <Library in current pdv project>  
-log <Log file path>  
-overwrite <optional parameter>
```

### **ECO Mode**

```
viewlogic2con  
-proj <path_for_proj_Lib>/<proj_file_name>.cpm  
-cdslib <path_for_cds.lib>/cds.lib
```

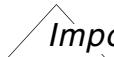
## Part Developer User Guide

### Part Developer Console Commands

---

```
-viewlogicdir <directory with ViewLogic files>
-package <ViewLogic file name without extension>
-ecolib <design_entry_HDL_library_name>
-ecocell <design_entry_HDL_cell_name>
[-IgnorePropDeletion]
-product pcb_librarian_expert
```

#### Parameter Description

Parameter	Description
proj	Absolute path to the project file.
cdslib	Absolute path to the <code>cds.lib</code> file.
lib	Destination Design Entry HDL library name.
cell	<Optional> Destination Design Entry HDL cell name. If this parameter is not specified, the ViewLogic part name is used to create the cell name.
viewlogicdir	Absolute path to the source directory where ViewLogic parts are stored.
package	The ViewLogic part name without extension.
overwrite	<Optional> Overwrites an existing cell.   <i>Important</i> An error message is displayed if this parameter is not specified and there already exists a cell with the same name in the destination library. The destination cell is not overwritten by default.
ecolib	Destination ECO library name.
ecocell	Destination ECO cell name.
IgnorePropDeletion	Retains the properties that exist in the destination cell but are not available in the source ECO file.

---

## Example

The following example creates a Design Entry HDL part, *s5*, from the ViewLogic package *s59201of1*:

```
viewlogic2con -proj d:/ulab/ulab_proj.cpm -cdslib d:/ulab/cds.lib -viewlogicdir  
d:/ulab/vlogic_dir -package s59201of1 -lib ulab_proj_lib -cell s5 -product  
pcb_librarian_expert
```

## xml2con

Creates a Design Entry HDL part from an XML datasheet.

## Usage

### ***Non-ECO Mode***

```
xml2con  
-cdslib <path_for_cds.lib>/cds.lib  
-lib <destination_design_entry_HDL_lib_name>  
-xmlfile <source_xml_file_path>/<xml_file_name>.xml  
[-overwrite]  
[-alias {AsCells|AsPrimitives}]  
-product pcb_librarian_expert
```

### ***ECO Mode***

```
xml2con  
-cdslib <path_for_cds.lib>/cds.lib  
-xmlfile <source_xml_file_path>/<xml_file_name>.xml  
-ecolib <name_of_eco_Lib>  
-ecocell <name_of_eco_Cell>  
[-IgnorePropDeletion]  
-product pcb_librarian_expert
```

## Parameter Description

Parameter	Description
cdslib	Absolute path to the <code>cds.lib</code> file.
lib	Destination Design Entry HDL library name.
xmlfile	Absolute path to the source XML file.
overwrite	<Optional> Overwrites an existing cell.
	 <b>Important</b> An error message is displayed if this parameter is not specified and there already exists a cell with the same name in the destination library. The destination cell is not overwritten by default.
alias {AsCells AsPrimitives} }	Creates separate cells or primitives depending on whether <code>alias</code> is specified with the <code>AsCells</code> or <code>AsPrimitives</code> option.
ecolib	Destination ECO library name.
ecoCell	Destination ECO cell name.
IgnorePropDeletion	Retains the properties that exist in the destination cell but are not available in the source ECO file.

## Example

The following example creates a Design Entry HDL part, `df`, from the XML datasheet `DF.xml`. If a part with the same name exists, it is overwritten.

```
xml2con -cdslib d:/ulab/cds.lib -lib ulab_proj_lib -xmlfile d:/ulab/DF.xml -  
overwrite -product pcb_librarian_expert
```

## csv2con

Creates a Design Entry HDL part from a CSV file.

## Usage

### ***Non-ECO Mode***

```
csv2con
  -proj <path_for_proj_Lib>/<proj_file_name>.cpm
  -cdslib <path_for_cds.lib>/cds.lib
  -lib <destination_design_entry_HDL_lib_name>
  [-cell <destination_design_entry_HDL_cell_name>]
  -csvfile <source_csv_file_path>/<csv_file_name>.csv
  [-overwrite]
  -proptemplatepath <path_to_template_file>
  -product pcb_librarian_expert
```

### ***ECO Mode***

```
csv2con
  -proj <path_for_proj_Lib>/<proj_file_name>.cpm
  -cdslib <path_for_cds.lib>/cds.lib
  -csvfile <source_csv_file_path>/<csv_file_name>.csv
  -ecolib <name_of_eco_Lib>
  -ecocell <name_of_eco_Cell>
  [-IgnorePropDeletion]
  -proptemplatepath <path_to_template_file>
  -product pcb_librarian_expert
```

## Parameter Description

---

Parameter	Description
proj	Absolute path to the project file.
cdslib	Absolute path to the <code>cds.lib</code> file.
lib	Destination Design Entry HDL library name.
cell	<Optional> Destination Design Entry HDL library name. If this parameter is not used, the csv file name is used as the cell name. For example, if the csv file name is <code>mypart.csv</code> , then the cell name will be <code>mypart</code> .
csvfile	Absolute path to the source CSV file.

## Part Developer User Guide

### Part Developer Console Commands

---

<b>Parameter</b>	<b>Description</b>
overwrite	<Optional> Overwrites an existing cell.
	 An error message is displayed if this parameter is not specified and there already exists a cell with the same name in the destination library. The destination cell is not overwritten by default.
ecolib	Destination ECO library name.
ecocell	Destination ECO cell name.
IgnorePropDeletion	Retains the properties that exist in the destination cell but are not available in the source ECO file.
proptemplatepath	Applies the properties listed in the specified properties template file (.sptpl), to the Design Entry HDL symbol being generated.

## Example

The following example creates a Design Entry HDL part from a CSV file. The project file and the cds.lib file are located at /hm/aoyon/csv\_import/outputs. The CSV file is located at /hm/aoyon/csv\_import/inputs/ls00.csv. The library in which the cell should be created is csv\_imp with the cell name as ls00.

```
csv2con -proj /hm/aoyon/csv_import/outputs/flow.cpm -cdslib /hm/aoyon/csv_import/outputs/cds.lib -csvfile /hm/aoyon/csv_import/inputs/ls00.csv -lib csv_imp -cell ls00 -product pcb_librarian_expert >>& ./partlog.log
```

## pinpak2con

Creates a Design Entry HDL part from an Si2 pinpak part.

## Usage

### ***Non-ECO Mode***

```
pinpak2con
    -proj <path_for_proj_Lib>/<proj_file_name>.cpm
    -cdslib <path_for_cds.lib>/cds.lib
```

## Part Developer User Guide

### Part Developer Console Commands

---

```
-lib <destination_design_entry_HDL_lib_name>
[-cell <destination_design_entry_HDL_cell_name>]
-pinpakfile <source_pinpak_xml_file_path>/<pinpak_xml_file_name>.xml
[-overwrite]
-product pcb_librarian_expert
```

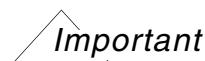
#### **ECO Mode**

```
pinpak2con
    -proj <path_for_proj_Lib>/<proj_file_name>.cpm
    -cdslib <path_for_cds.lib>/cds.lib
    -pinpakfile <source_pinpak_xml_file_path>/<pinpak_xml_file_name>.xml
    -ecolib <name_of_eco_Lib>
    -ecocell <name_of_eco_Cell>
    [-IgnorePropDeletion]
    -product pcb_librarian_expert
```

#### **Parameter Description**

---

Parameter	Description
proj	Absolute path to the project file.
cdslib	Absolute path to the <code>cds.lib</code> file.
lib	Destination Design Entry HDL library name.
cell	<Optional> Destination Design Entry HDL library name. If this parameter is not used, the pinpak file name is used as the cell name.
pinpakfile	Absolute path to the source PinPak file.
overwrite	<Optional> Overwrites an existing cell.



An error message is displayed if this parameter is not specified and there already exists a cell with the same name in the destination library. The destination cell is not overwritten by default.

ecolib      Destination ECO library name.

## Part Developer User Guide

### Part Developer Console Commands

---

<b>Parameter</b>	<b>Description</b>
ecocell	Destination ECO cell name.
IgnorePropDeletion	Retains the properties that exist in the destination cell but are not available in the source ECO file.

---

## Example

The following example creates a Design Entry HDL part, *mipinpak*, from a Si2 pinpak part datasheet, *mipinpak.xml*. If a part with the same name exists, it is overwritten.

```
pinpak2con -proj d:/ulab/ulab_proj.cpm -cdslib d:/ulab/cds.lib -lib ulab_proj_lib  
-pinpakfile d:/ulab/mipinpak.xml -overwrite -product pcb_librarian_expert
```

## ptm2con

Creates a Design Entry HDL part from a Synopsys PTM part.

## Usage

### **Non-ECO Mode**

```
ptm2con  
-proj <path_for_proj_Lib>/<proj_file_name>.cpm  
-cdslib <path_for_cds.lib>/cds.lib  
-lib <destination_design_entry_HDL_lib_name>  
[-cell <destination_design_entry_HDL_cell_name>]  
-ptmfile <source_ptm_file_path>/<ptm_file_name>.ptm  
[-overwrite]  
-product pcb_librarian_expert
```

### **ECO Mode**

```
ptm2con  
-proj <path_for_proj_Lib>/<proj_file_name>.cpm  
-cdslib <path_for_cds.lib>/cds.lib  
-ptmfile <source_ptm_file_path>/<ptm_file_name>.ptm  
-ecolib <name_of_eco_Lib>  
-ecocell <name_of_eco_Cell>
```

## Part Developer User Guide

### Part Developer Console Commands

---

```
[-IgnorePropDeletion]
-product pcb_librarian_expert
```

#### Parameter Description

---

Parameter	Description
proj	Absolute path to the project file.
cdslib	Absolute path to the <code>cds.lib</code> file.
lib	Destination Design Entry HDL library name.
cell	<Optional> Destination Design Entry HDL library name. If this parameter is not used, the pinpak file name is used as the cell name.
ptmfile	Absolute path to the source PtM file.
overwrite	<Optional> Overwrites an existing cell.
 <i>Important</i>	
An error message is displayed if this parameter is not specified and there already exists a cell with the same name in the destination library. The destination cell is not overwritten by default.	
ecolib	Destination ECO library name.
ecocell	Destination ECO cell name.
IgnorePropDeletion	Retains the properties that exist in the destination cell but are not available in the source ECO file.

---

#### Example

The following example creates a Design Entry HDL part, `t1174`, from a PTM part, `TTL174.ptm`:

```
ptm2con -proj d:/ulab/ulab_proj.cpm -cdslib d:/ulab/cds.lib -lib ulab_proj_lib -
ptmfile d:/ulab/TTL174.ptm -product pcb_librarian_expert
```

## **vhdl2con**

Creates a Design Entry HDL part from a VHDL model.

### **Usage**

```
vhdl2con
  -proj <path_for_proj_Lib>/<proj_file_name>.cpm
  -cdslib <path_for_cds.lib>/cds.lib
  -lib <destination_design_entry_HDL_lib_name>
  [-cell <destination_design_entry_HDL_cell_name>]
  -vhdlfile <source_vhdl_file_path>/<vhdl_file_name>.vhd
  [-overwrite]
  -product pcb_librarian_expert
```

### **Parameter Description**

<b>Parameter</b>	<b>Description</b>
proj	Absolute path to the project file.
cdslib	Absolute path to the <code>cds.lib</code> file.
lib	Destination Design Entry HDL library name.
cell	<Optional> Destination Design Entry HDL library name. If this parameter is not used, the pinpak file name is used as the cell name.
vhdlfile	Absolute path to the source VHDL file.
overwrite	<Optional> Overwrites an existing cell.



An error message is displayed if this parameter is not specified and there already exists a cell with the same name in the destination library. The destination cell is not overwritten by default.

## Example

The following example creates a Design Entry HDL part, 74ac74, from the VHDL model 74ac74.vhd:

```
vhdl2con -proj d:/ulab/ulab_proj.cpm -cdslib d:/ulab/cds.lib -lib ulab_proj_lib -vhdlfile d:/ulab/74ac74.vhd -product pcb_librarian_expert
```

## verilog2con

Creates a Design Entry HDL part from a Verilog model.

### Usage

```
verilog2con  
-proj <path_for_proj_Lib>/<proj_file_name>.cpm  
-cdslib <path_for_cds.lib>/cds.lib  
-lib <destination_design_entry_HDL_lib_name>  
[-cell <destination_design_entry_HDL_cell_name>]  
-verilogfile <source_verilog_file_path>/<verilog_file_name>.v  
[-overwrite]  
-product pcb_librarian_expert
```

### Parameter Description

---

Parameter	Description
proj	Absolute path to the project file.
cdslib	Absolute path to the cds.lib file.
lib	Destination Design Entry HDL library name.
cell	<Optional> Destination Design Entry HDL library name. If this parameter is not used, the pinpak file name is used as the cell name.
verilogfile	Absolute path to the source Verilog file.

## Part Developer User Guide

### Part Developer Console Commands

---

<b>Parameter</b>	<b>Description</b>
overwrite	<Optional> Overwrites an existing cell.   An error message is displayed if this parameter is not specified and there already exists a cell with the same name in the destination library. The destination cell is not overwritten by default.

## Example

The following example creates a Design Entry HDL part, `27s181`, from the Verilog file `27s181.v`:

```
verilog2con -proj d:/ulab/ulab_proj.cpm -cdslib d:/ulab/cds.lib -verilogfile d:/ulab/27s181.v -lib vlog_imp -cell 27s181 -product pcb_librarian_expert
```

## con2xml

Creates an XML datasheet from a Design Entry HDL part.

## Usage

```
con2xml  
-cdslib <path_for_cds.lib>/cds.lib  
-lib <source_design_entry_HDL_lib_name>  
-cell <source_design_entry_HDL_cell_name>  
-package <package_name>  
-xmldir <destination_dir_path>  
[-overwrite]  
-product pcb_librarian_expert
```

## Parameter Description

Parameter	Description
cdslib	Absolute path to the <code>cds.lib</code> file.
lib	Source Design Entry HDL library name.
cell	Source Design Entry HDL library name. If this parameter is not used, the pinpak file name is used as the cell name.
package	Name of the package to be exported.
xmldir	Absolute path to the destination directory.
overwrite	<Optional> Overwrites an existing cell.



An error message is displayed if this parameter is not specified and there already exists a cell with the same name in the destination library. The destination cell is not overwritten by default.

## Example

The following example creates an XML datasheet, `S5920.xml`, at `d:/ulab` from the Design Entry HDL part `s5920`:

```
con2xml -cdslib d:/ulab/cds.lib -lib ulab_proj_lib -cell s5920 -package S5920 -  
xmldir d:/ulab -product pcb_librarian_expert
```

## con2csv

Creates a CSV file from a Design Entry HDL part.

## Usage

```
con2csv  
-cdslib <path_for_cds.lib>/cds.lib  
-lib <source_design_entry_HDL_lib_name>  
-cell <source_design_entry_HDL_cell_name>
```

```
-package <package_name>
-csvdir <destination_dir_path>
[-overwrite]
-product pcb_librarian_expert
```

## Parameter Description

---

Parameter	Description
cdslib	Absolute path to the <code>cds.lib</code> file.
lib	Source Design Entry HDL library name.
cell	Source Design Entry HDL library name. If this parameter is not used, the pinpak file name is used as the cell name.
package	Name of the package to be exported.
csvdir	Absolute path to the destination directory.
overwrite	<Optional> Overwrites an existing cell.



An error message is displayed if this parameter is not specified and there already exists a cell with the same name in the destination library. The destination cell is not overwritten by default.

---

## Example

The following example creates a CSV datasheet, `s5920.csv`, from the Design Entry HDL part `s5920`.

```
con2csv -cdslib d:/ulab/cds.lib -lib ulab_proj_lib -cell s5920 -package S5920 -
          csvdir d:/ulab -product pcb_librarian_expert
```

## cap2con

Creates a Design Entry HDL part or library from a Capture part or library.

## Usage

### **Non-ECO Mode**

```
cap2con
  [-proj <path_for_proj_Lib>/<proj_file_name>.cpm]
  -cdslib <path_for_cds.lib>/cds.lib
  -conceptlib <destination_design_entry_HDL_lib_name>
  -olbpath <source_capture_lib_path>/<library_name>.olb
  [-part <capture_part_name>]
  [-alias {ascells|asprimitives}]
  [-overwrite]
  -product pcb_librarian_expert
```

### **ECO Mode**

```
cap2con
  [-proj <path_for_proj_Lib>/<proj_file_name>.cpm]
  -cdslib <path_for_cds.lib>/cds.lib
  -olbpath <source_capture_lib_path>/<library_name>.olb
  -part <capture_part_name>
  -ecolib <name_of_eco_Lib>
  -ecocell <name_of_eco_Cell>
  [-IgnorePropDeletion]
  -product pcb_librarian_expert
```

## Parameter Description

---

Parameter	Description
proj	<Optional> Absolute path to the project file.
cdslib	Absolute path to the <code>cds.lib</code> file.
conceptlib	Destination Design Entry HDL library.
olbpath	Absolute path to the source Capture library.
part	<Optional> The part name to be converted or ECOed. If the part name option is not specified in non-ECO mode, all parts in the library are converted as Design Entry HDL cells. This option is mandatory in ECO mode.

## Part Developer User Guide

### Part Developer Console Commands

---

<b>Parameter</b>	<b>Description</b>
overwrite	<Optional> Overwrites an existing cell.
	 <b>Important</b> An error message is displayed if this parameter is not specified and there already exists a cell with the same name in the destination library. The destination cell is not overwritten by default.
alias {AsCells AsPrimitives }	Creates separate cells or primitives depending on whether alias is specified with the AsCells or AsPrimitives option.  <b>Note:</b> If this parameter is not specified, only the master component is imported.
ecolib	Destination ECO library name.
ecocell	Destination ECO cell name.
IgnorePropDeletion	Retains the properties that exist in the destination cell but are not available in the source ECO file.

---

## Example

The following example converts all the parts in the source Capture library, basic.olb, to separate primitives in the destination Design Entry HDL library, mylib.

```
cap2con -proj D:/cap2con/cap2con.cpm -olbpath D:/capture/basic.olb -alias  
asprimitives -cdslib D:/cap2con/cds.lib -conceptlib mylib -product  
pcb_librarian_expert
```

## con2cap

Creates a Capture part from a Design Entry HDL part.

## Usage

```
con2cap  
-cdslib <path_for_cds.lib>/cds.lib
```

## Part Developer User Guide

### Part Developer Console Commands

---

```
-conceptlib <source_design_entry_HDL_lib_name>
-cell <source_design_entry_HDL_cell_name>
-olbpath <destination_Capture_library_path>/<library_name>.olb
-package <package_name_in_cell>
[-symbolist <comma_separated_list_of_symbols>]
[-usepinnames]
-product pcb_librarian_expert
```

#### Parameter Description

Parameter	Description
cdslib	Absolute path to the <code>cds.lib</code> file.
conceptlib	Source Design Entry HDL library.
cell	Source Design Entry HDL cell name.
olbpath	Absolute path to the destination Capture library. If the library does not exist, then it is created. Otherwise, parts are added to the Capture library.
package	Name of the package to be converted.
symbolist	<Optional> The comma-separated list of symbols to be converted.
usepinnames	Makes Design Entry HDL pin names as Capture pin names. If this parameter is not specified, pin text is used to determine the Capture pin names.

#### Example

The following example creates a Capture part, `PROJ.OLB`, from the Design Entry HDL part `ss59`. The Capture part contains symbol information for the specified symbols, `sym_1` and `sym_2`.

```
con2cap -cdslib d:/ulab/cds.lib -conceptlib ulab_proj_lib -cell ss59 -olbpath d:/
    ulab/proj.olb -package S5920 -symbolist sym_1,sym_2 -product
        pcb_librarian_expert
```

## cap2xml

Creates an XML datasheet from a Capture part.

### Usage

```
cap2xml
  -olbpath <source_capture_lib_path>/<library_name>.olb
  [-part <capture_part_name>]
  -xmldir <destination_dir_path>
  [-overwrite]
  -product pcb_librarian_expert
```

### Parameter Description

Parameter	Description
olbpath	Absolute path to the source Capture library.
part	<Optional> Name of the part to be converted. If the -part parameter is not specified, all the parts in the library will get translated.
xmldir	Absolute path to the output directory.
overwrite	<Optional> Overwrites an existing cell.



An error message is displayed if this parameter is not specified and there already exists a cell with the same name in the destination library. The destination cell is not overwritten by default.

### Example

The following example creates an XML datasheet, *S5920.xml*, from a Capture part, *PROJ.OLB*. The name of the datasheet is derived from the package name.

```
cap2xml -olbpath d:/ulab/PROJ.OLB -xmldir d:/ulab -product pcb_librarian_expert
```

## xml2cap

Creates a Capture part from an XML datasheet.

### Usage

```
xml2cap
  -xmlfile <source_xml_file_path>/<xml_file_name>.xml
  -olbpath <destination_Capture_library_path>/<library_name>.olb
  [-package <package_name_in_cell>]
  [-symbollist <comma_separated_list_of_symbols>]
  [-usepinnames]
  -product pcb_librarian_expert
```

### Parameter Description

Parameter	Description
xmlfile	Absolute path to the source xml file.
olbpath	Absolute path to the destination Capture library.
package	Name of the package to be converted.
symbollist	<Optional> The comma-separated list of symbols to be converted.
usepinnames	Makes Design Entry HDL pin names as Capture pin names. If this parameter is not specified, pin text is used to determine the Capture pin names.

### Example

The following example creates a Capture part, *S5920.OLB*, from the XML datasheet, *S5920.xml*.

```
xml2cap -xmlfile d:/ulab/s5920.xml -olbpath d:/ulab/S5920.olb -product
  pcb_librarian_expert
```

## apd2con

Creates a Design Entry HDL part from an APD component.

### Usage

#### ***Non-ECO Mode***

```
apd2con
    -proj <project_file>
    -cdslib <cds.lib path>
    -apddir <directory with APD files>
    -lib <concept library name>
    [-cell <concept cell name>]
    [-overwrite]
    -product pcb_librarian_expert
```

#### ***ECO Mode***

```
apd2con
    -proj <project_file>
    -cdslib <cds.lib path>
    -apddir <directory with APD files>
    [-ecopindelayonly]
    -ecolib <concept library name>
    -ecocell <concept cell name>
    [-IgnorePropDeletion]
    -product pcb_librarian_expert
```

### Parameter Description

---

Parameter	Description
proj	Absolute path to the project file.
cdslib	Absolute path to the <code>cds.lib</code> file.
apddir	Absolute path to the APD-generated component directory.
lib	Destination Design Entry HDL library.

## Part Developer User Guide

### Part Developer Console Commands

---

<b>Parameter</b>	<b>Description</b>
cell	Destination Design Entry HDL cell name.
overwrite	<Optional> Overwrites an existing cell.
	 <b>Important</b> An error message is displayed if this parameter is not specified and there already exists a cell with the same name in the destination library. The destination cell is not overwritten by default.
ecopindelayonly	ECOs only the pin delay values from the source APD component into the destination Design Entry HDL cell. This is done on the basis of physical pin numbers.
ecolib	Destination ECO library name.
ecocell	Destination ECO cell name.
IgnorePropDeletion	Retains the properties that exist in the destination cell but are not available in the source ECO file.

---

## Example

The following example creates a Design Entry HDL part from an APD component:

```
apd2con -proj d:/ulab/ulab_proj.cpm -cdslib d:/ulab/cds.lib -apddir d:/ulab/  
apd_import -lib ulab_proj_lib -product pcb_librarian_expert
```

## dml2con

Creates a Design Entry HDL part from a DML model.

## Usage

### **Non-ECO Mode**

```
dml2con
```

## Part Developer User Guide

### Part Developer Console Commands

---

```
-proj <path_for_proj_Lib>/<proj_file_name>.cpm
-cdslib <path_for_cds.lib>/cds.lib
-lib <destination_concept_lib_name>
[-cell <destination_concept_cell_name>]
-dmlfile <source_dml_file_path>/<dml_file_name>.dml
-device <source_dml_device_name>/<dml_device_name>
[-overwrite]
-product pcb_librarian_expert
```

### ***ECO Mode***

dml2con

```
-proj <path_for_proj_Lib>/<proj_file_name>.cpm
-cdslib <path_for_cds.lib>/cds.lib
-dmlfile <source_dml_file_path>/<dml_file_name>.dml
-device <source_dml_device_name>/<dml_device_name>
-ecolib <name_of_eco_Lib>
-ecocell <name_of_eco_Cell>
[-IgnorePropDeletion] [-IgnoreGraphicMod]
-product pcb_librarian_expert
```

## Part Developer User Guide

### Part Developer Console Commands

---

#### Parameter Description

Parameter	Description
proj	Absolute path to the project file.
cdslib	Absolute path to the <code>cds.lib</code> file.
lib	Destination Design Entry HDL library.
cell	<Optional> Destination Design Entry HDL cell name.
dmlfile	Absolute path to the source DML file.
device	Name of the device to be converted.
overwrite	<Optional> Overwrites an existing cell.

 *Important*

An error message is displayed if this parameter is not specified and there already exists a cell with the same name in the destination library. The destination cell is not overwritten by default.

ecolib	Destination ECO library name.
ecocell	Destination ECO cell name.
IgnorePropDeletion	Retains the properties that exist in the destination cell but are not available in the source ECO file.
IgnoreGraphicMod	Retains the original symbol in the destination cell.

#### Example

The following example creates a Design Entry HDL part from a DML model file, `41374k.dml`:

```
dml2con -proj d:/ulab/ulab_proj.cpm -cdslib d:/ulab/cds.lib -lib ulab_proj_lib -  
dmlfile d:/ulab/41374k.dml -product pcb_librarian_expert
```

## ibis2con

Creates a Design Entry HDL part from an IBIS model.

## Usage

### ***Non-ECO Mode***

ibis2con

```
-proj <path_for_proj_Lib>/<proj_file_name>.cpm
-cdllib <path_for_cds.lib>/cds.lib
-lib <destination_concept_lib_name>
[-cell <destination_concept_cell_name>]
-ibisfile <source_ibis_file_path>/<ibis_file_name>.ibs
-device <source_ibs_device_name>/<ibs_device_name>
[-overwrite]
-product pcb_librarian_expert
```

### ***ECO Mode***

ibis2con

```
-proj <path_for_proj_Lib>/<proj_file_name>.cpm
-cdllib <path_for_cds.lib>/cds.lib
-ibisfile <source_ibis_file_path>/<ibis_file_name>.ibs
-device <source_ibs_device_name>/<ibs_device_name>
-ecolib <name_of_eco_Lib>
-ecocell <name_of_eco_Cell>
[-IgnorePropDeletion] [-IgnoreGraphicMod]
-product pcb_librarian_expert
```

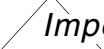
## Part Developer User Guide

### Part Developer Console Commands

---

#### Parameter Description

Parameter	Description
proj	Absolute path to the project file.
cdslib	Absolute path to the <code>cds.lib</code> file.
lib	Destination Design Entry HDL library.
cell	<Optional> Destination Design Entry HDL cell name.
ibisfile	Absolute path to the source DML file.
device	Name of the device to be converted.
overwrite	<Optional> Overwrites an existing cell.

 *Important*

An error message is displayed if this parameter is not specified and there already exists a cell with the same name in the destination library. The destination cell is not overwritten by default.

ecolib	Destination ECO library name.
ecocell	Destination ECO cell name.
IgnorePropDeletion	Retains the properties that exist in the destination cell but are not available in the source ECO file.
IgnoreGraphicMod	Retains the original symbol in the destination cell.

#### Example

The following example creates a Design Entry HDL part, `c20ke`, from the IBIS model `c20ke.ibs`.

```
ibis2con -proj d:/ulab/ulab_proj.cpm -cdslib d:/ulab/cds.lib -lib ulab_proj_lib -  
ibisfile d:/ulab/ c20ke.ibs -product pcb_librarian_expert
```

## **lib2cellptf**

Converts a library-level part table file to cell-level part table files.

### **Usage**

```
lib2cellptf
  -cdslib <path_for_cds.lib>/cds.lib
  -ptffile <path_for_lib_level_ptf_file>/<ptf_file_name>.ptf | -proj
    <path_for_proj_Lib>/<proj_file_name>.cpm
  [-lib <destination library>]
  [-cell <destination cell>]
  { [-ptfnameascellname] [-overwrite] }
  -product pcb_librarian_expert
```

## Parameter Description

Parameter	Description
cdslib	Absolute path to the <code>cds.lib</code> file.
ptffile	Absolute path to the library-level part table file.
proj	Absolute path to the project file.
lib	<Optional> Destination Design Entry HDL library.
cell	<Optional> Destination Design Entry HDL cell name.
ptfnameascellname	Use the ptf name as the cell name.
overwrite	<Optional> Overwrites the existing ptf.



An error message is displayed if this parameter is not specified and there already exists a cell with the same name in the destination library. The destination cell is not overwritten by default. This option is valid only in combination with the `ptfnameascellname` option.

---

## Example

The following example creates PTF files for all the cells of the libraries defined in `cds.lib`:

```
lib2cellptf -cdslib d:/ulab/cds.lib -ptffile d:/ulab/lib.ptf -product  
pcb_librarian_expert
```

## fpga2con

Creates a Design Entry HDL part from an FPGA component.

### Usage

#### ***Non-ECO Mode***

```
fpga2con
```

## Part Developer User Guide

### Part Developer Console Commands

---

```
-proj <project_file>
-cdslib <cds.lib path>
-vendor actel|altera|xilinx
-pinfile|padfile <path to .pne/.pin/.pad file>
-pnr maxplusII|quartusII (For vendor=altera only)
-pkgfile <path to .pkg file> (For vendor=Actel only)
-pgafile <path to .pga.pin file> (For vendor=Actel only)
[-hdlfile <path to vhdl|vlog files>]
[-sdffile <path to sdf file>]
[-stdlib <std component lib name> -stdcell <std component cell name>]
-lib <concept library name>
[-cell <concept cell name>]
[-overwrite]
-product pcb_librarian_expert
```

#### ***ECO Mode***

```
fpga2con
-proj <project_file>
-cdslib <cds.lib path>
-vendor actel|altera|xilinx
-pinfile|padfile <path to .pne/.pin/.pad file>
-pnr maxplusII/quartusII (For vendor=altera only)
-pkgfile <path to .pkg file> (For vendor=Actel only)
-pgafile <path to .pga.pin file> (For vendor=Actel only)
[-hdlfile <path to vhdl/vlog files>] [-sdffile <path to sdf file>]
[-stdlib <std component lib name> -stdcell <std component cell name>]
-ecolib <concept library name> -ecocell <concept cell name>
[-IgnorePropDeletion]
[-IgnoreGraphicMod]
-product pcb_librarian_expert
```

## Part Developer User Guide

### Part Developer Console Commands

---

#### Parameter Description

Parameter	Description
proj	Absolute path to the project file.
cdslib	Absolute path to the <code>cds.lib</code> file.
lib	Destination Design Entry HDL library.
cell	<Optional> Destination Design Entry HDL cell name.
vendor	Name of the FPGA vendor. The possible choices are Actel, Altera, and XILINX.
pinfile padfile	Absolute path to the <code>.pne</code> , <code>.pin</code> , or <code>.pad</code> file.
pnr	To be used when the vendor is Altera. The possible choices are <code>maxplusII</code> and <code>quartusII</code> .
pkgfile	Absolute path to the <code>.pkg</code> file. This is used when the vendor is Actel.
pgofile	Absolute path to the <code>.pga</code> file. This is used when the vendor is Actel.
hdlfile	Absolute path to the Verilog or VHDL file.
sdffile	Absolute path to the <code>sdf</code> file.
stdlib	Standard component library name.
stdcell	Standard component cell name.
overwrite	<Optional> Overwrites an existing cell.

 *Important*

An error message is displayed if this parameter is not specified and there already exists a cell with the same name in the destination library. The destination cell is not overwritten by default.

ecolib	Destination ECO library name.
ecoceil	Destination ECO cell name.
IgnorePropDeletion	Retains the properties that exist in the destination cell but are not available in the source ECO file.

## Part Developer User Guide

### Part Developer Console Commands

---

<b>Parameter</b>	<b>Description</b>
IgnoreGraphicMod	Retains the original symbol in the destination cell.

## Example

The following example creates a Design Entry HDL part from an AlteraMaxPlusII place-and-route file, altchip.pin.

```
fpga2con -proj d:/ulab/ulab_proj.cpm -cdslib d:/ulab/cds.lib -vendor altera -  
pinfile d:/ulab/altchip.pin -pnr maxplusII -lib ulab_proj_lib -product  
pcb_librarian_expert
```

## allegro2con

Creates a Design Entry HDL part from an Allegro footprint.

## Usage

### ***Non-ECO Mode***

```
allegro2con  
-proj <path_for_proj_Lib>/<proj_file_name>.cpm  
-cdslib <path_for_cds.lib>/cds.lib  
-lib <destination_design_entry_HDL_lib_name>  
[-cell <destination_design_entry_HDL_cell_name>]  
-ftprint <footprint name present in PSMPATH>  
[-overwrite]  
-product pcb_librarian_expert
```

### ***ECO Mode***

```
allegro2con  
-proj <path_for_proj_Lib>/<proj_file_name>.cpm  
-cdslib <path_for_cds.lib>/cds.lib  
-ecolib <eco_lib_name>  
-ecocell <eco_cell_name>  
[-IgnorePropDeletion]  
[-IgnoreGraphicMod]  
-ftprint <footprint name present in PSMPATH>
```

## Part Developer User Guide

### Part Developer Console Commands

---

```
-product pcb_librarian_expert
```

#### Parameter Description

Parameter	Description
proj	Absolute path to the project file.
cdslib	Absolute path to the <code>cds.lib</code> file.
lib	Destination Design Entry HDL library name.
cell	<Optional> Destination Design Entry HDL library name. If this parameter is not used, the footprint name is used as the cell name.
ftprint	Name of the footprint that is present in PSMPATH.
overwrite	<Optional> Overwrites an existing cell.
 <i>Important</i>	
An error message is displayed if this parameter is not specified and there already exists a cell with the same name in the destination library. The destination cell is not overwritten by default.	
ecolib	Destination ECO library name.
ecocell	Destination ECO cell name.
IgnorePropDeletion	Retains the properties that exist in the destination cell but are not available in the source ECO file.
IgnoreGraphicMod	Retains the original symbol in the destination cell.

#### Example

The following example creates a Design Entry HDL part from a footprint, conn50:

```
allegro2con -proj d:/ulab/ulab_proj.cpm -cdslib d:/ulab/cds.lib -lib ulab_proj_lib  
-ftprint conn50 -product pcb_librarian_expert
```

## **text2con**

Creates a Design Entry HDL part from a text file and a profile file.

### **Usage**

#### ***Non-ECO Mode***

```
text2con
  -proj <path_for_proj_Lib>/<proj_file_name>.cpm
  -cdslib <path_for_cds.lib>/cds.lib
  -lib <destination_design_entry_HDL_lib_name>
  [-cell <destination_design_entry_HDL_cell_name>]
  -textfile <source_text_file_path>/<text_file_name>
  -profilefile <profile_file_path>/<profile_file_name>.prf
  [-footprint <footprint name present in PSMPATH>]
  [-overwrite]
  -product pcb_librarian_expert
```

#### ***ECO Mode***

```
text2con
  -proj <path_for_proj_Lib>/<proj_file_name>.cpm
  -cdslib <path_for_cds.lib>/cds.lib
  -textfile <source_text_file_path>/<text_file_name>
  -profilefile <profile_file_path>/<profile_file_name>.prf
  [-footprint <footprint name present in PSMPATH>]
  -ecolib <name_of_eco_Lib>
  -ecocell <name_of_eco_Cell>
  [-IgnorePropDeletion]
  [-IgnoreGraphicMod]
  -product pcb_librarian_expert
```

## Parameter Description

Parameter	Description
proj	Absolute path to the project file.
cdslib	Absolute path to the <code>cds.lib</code> file.
lib	Destination Design Entry HDL library name.
cell	<Optional> Destination Design Entry HDL library name. If this parameter is not used, the text file name is used as the cell name. For example, if the text file name is <code>mypart.txt</code> , then the cell name will be <code>mypart</code> .
textfile	Absolute path to the source text file.
profilefile	Absolute path to the profile file.
footprint	Footprint value from PSMPATH to write as JEDEC_TYPE in chips for the imported part.
overwrite	<Optional> Overwrites an existing cell.
 <b>Important</b>	
An error message is displayed if this parameter is not specified and there already exists a cell with the same name in the destination library. The destination cell is not overwritten by default.	
ecolib	Destination ECO library name.
ecoCell	Destination ECO cell name.
IgnorePropDeletion	Retains the properties that exist in the destination cell but are not available in the source ECO file.
IgnoreGraphicMod	Retains the symbol graphics that exist in the destination cell.

---

## Example

The following example creates a Design Entry HDL part, `transceiver`, from the text file `transceiver.txt` by using a profile file, `text_profile.prf`.

## **Part Developer User Guide**

### Part Developer Console Commands

---

```
text2con -proj d:/ulab/ulab_proj.cpm -cdslib d:/ulab/cds.lib -lib ulab_proj_lib -  
    textfile d:/ulab/transceiver.txt -profilefile d:/ulab/text_profile.prf -  
    product pcb_librarian_expert
```

---

## Dialog Box Help

---

### Open Project

The Open Project dialog box appears when you choose the *Open Project* option from the *File* menu. This dialog box enables you to open a project in Part Developer. It has the following elements:

#### **Project**

Enables you to specify the project to be opened.

#### **Browse**

Enables you to browse to the location where the project file is stored.

### New Cell

The New Cell dialog box appears when you choose the *New – Cell* option from the *File* menu. This dialog box enables you to create a new part in a specific library. It has the following elements:

#### **Library**

Displays the list of libraries in the project. The library list is generated from the library entries in the `cds.lib` file. You need to select a library in which to create the new part.

#### **Cell**

Enables you to enter the name of the part that is being created.

## Open Cell

The Open Cell dialog box appears when you choose the *Open – Cell* option from the *File* menu. This dialog box enables you to open a library part. It has the following elements:

### Library

Displays the list of libraries in the project. The library list is generated from the library entries in the `cds.lib` file. You need to select a library in which to create the new part.

### Cell

Enables you to select the part that you want to open. On selecting the part and clicking OK, the part gets loaded in the Cell Editor.

## Edit Functions

The Edit Functions dialog box appears when the *Functions/Slots* button is clicked on the Package Pin page of the Package Editor. The Edit Functions dialog box enables you to add or remove slots in a package and distribute pins across slots. It has the following elements:

### Functions

Displays the number of slots in a package. For example, for LS241, the *Functions* column displays eight rows (S1 to S8) with each row representing a slot.

### SPLIT\_INST\_GROUP

The `SPLIT_INST_GROUP` property is used for split parts. The value enables Part Developer to determine which slots of a split part combine to form one logical slot.

### Replicate value of `SPLIT_INST_GROUP` property for added section

Enables you to replicate the value of `SPLIT_INST_GROUP` in the slots that are added by clicking *Add*.

**Note:** The replicated value for the new slot is always a copy of the last available `SPLIT_INST_GROUP` value. For example, suppose there are four slots—slots S1 and S2 have a `SPLIT_INST_GROUP` value of 1 and slots S3 and S4 have a `SPLIT_INST_GROUP`

value of 2. If you select the *Replicate value of SPLIT\_INST\_GROUP property for added section* option and add a new slot, the new slot, S5, will have a `SPLIT_INST_GROUP` value of 2.

### Add

Enables you to add slots to the package

### Delete

Enables you to delete the selected slot from the package. On deleting a slot, the pin mapping information for that slot is lost from the package and the physical pins appear as unmapped in the *Physical Pins* grid.

### Distribute Pins

Displays the Distribute Pins dialog box. You use the Distribute Pins dialog box to determine which logical pins are present in which slots.

## Distribute Pins

The Distribute Pins dialog box appears when you click the *Distribute Pins* button in the Edit Functions dialog box. The Distribute Pins dialog box enables you to distribute the logical pins across the slots of a package. It has the following elements:

### Pin Name

Displays the name of the pins in the package

### S1..Sn (n)

Represent the number of slots in a package. The number in parentheses after the slot number represents the number of pins present in the slot. The check boxes under these columns determine whether a particular logical pin is present in the slot. To make a pin available in a particular slot, select the check box under the slot for the pin. For example, consider a two-slot part with two pins A and B. Suppose pin A exists in slot 1 and pin B in slot 2. Now, to add pin A to the second slot, select the check box under the S2 column for pin A.

**Note:** Part Developer provides the copy-paste functionality, which enables you to replicate the status of pin availability across slots. For example, consider a package, my\_part\_dip, with three pins A, B, and C. Pins A and C are present in slot S1 and pin B in slot S2. Now, you add a new slot S3 and want to put pins in slot S1 to it. To do so, drag and select the cells between pins A and C under slot S1, press Ctrl + C, select the first cell under S3, and press Ctrl + V.

### **Pin on one symbol only (For Split Part Only)**

Ensures that a pin is present in one and only one slot for a split part. This ensures that a pin is present in only one symbol (because each slot is represented by a unique symbol).

### **All bus bits on same symbol (For Split Part Only)**

Ensures that for a split part, all the bits of a vector pin are present in one and only one symbol.

## **Add Pin**

The Add Pin dialog box appears when you choose the *Pins – Add* option from the Symbol Editor or the Package Editor. This dialog box enables you to add all the pins of the part. See [Adding Logical Pins on page 112](#) for details.

The Add Pin dialog box is divided into two group boxes: *Add New Pins* and a grid that shows all the pins for the part. You enter the pins that you want to add to any package and/or symbol of the part through the *Add New Pins* section. These pins appear in the pins grid when you click *Add*. You can then choose the pins that you want to add to a package/ symbol from the list of available pins.

The *Add New Pins* group box has the following elements:

### **Scalar**

Enables you to enter scalar pin names such as A, B, C, or A1, A2 and so on. To enter scalar pin names, you need to specify the following:

<b>Prefix</b>	The prefix for scalar pin names. For example, if you want to add pin names A1B..A10B, the value in the <i>Prefix</i> field will be A.
---------------	---

## Part Developer User Guide

### Dialog Box Help

---

From	The starting value of the pin range. For example, if you want to add pins A1B..A10B, the value in the <i>From</i> field will be 1.
To	The end value of the pin range. For example, if you want to add pins A1B..A10B, the value in the <i>To</i> field will be 10.
Suffix	The suffix for scalar pin names. For example, if you want to add pin names A1B..A10B, the value in the <i>Suffix</i> field will be B.

### **Vector**

Enables you to enter pin names for a vector pin, such as A<1..10>. To enter pin names for vector pins, you need to specify the following:

Base Name	The base name of the vector pin. For example, if you want to add vector pin A<1..10>, the value in the <i>Base Name</i> field will be A.
MSB	The most significant bit of the vector pin. For a vector pin A<1..10>, the <i>MSB</i> value will be 10.
LSB	The least significant bit of the vector pin. For a vector pin A<1..10>, the <i>LSB</i> value will be 1.

### **Sizeable**

Enables you to enter sizeable pins. This option is available when you add pins from within the Symbol Editor.

### **SizeString**

Determines the notation when creating the sizeable pin. The two notations are <SIZE..1> and <SIZE-1..0>. You need to choose one of the two when specifying symbol pins.

### Type

Determines the pin type of the pins being added. The possible pin types are ANALOG, BIDIR, GROUND, INPUT, NC, OC, OC\_BIDIR, OE, OE\_BIDIR, OUTPUT, POWER, TS, TS\_BIDIR, and UNSPEC.

### Location

Determines where in the symbol the pin will be added. The possible location values are the left, right, top, and bottom of a symbol outline.

### Shape

Displays the name of the shape that is to be attached to the pin.

### Load

Enables you to specify the load values for the particular pin types. You can specify the following load values:

- Input low
- Input high
- Output low
- Output high

These fields are enabled only for those pin types for which load values are applicable, such as INPUT and OC. The default values for these fields are taken from the values specified in Setup. If required, you can modify the load values.

### Checks

Enables you to determine the checks that will be run on the pins when Rules Checker is run on the part. The types of checks that can be set are:

- Direction
- Assert
- Output
- Load

- IO
- Unknown loading

### **Add**

Adds the pin information to the pin grid at the bottom of the Add Pin dialog box

### **Pins Grid**

Displays those pins that are not present in the package/symbol from which you invoked the Add Pin dialog box but are present in some other package/symbol. You can choose to add such pins to the current package/symbol. It has the following columns:

#### **Select**

Enables you to add a pin from a package/symbol by selecting/deselecting the check box next to the pin.

#### **Name**

Displays the pin names. This field is non-editable.

#### **MSB**

Displays the most significant bit of a vector pin. This field is non-editable.

#### **LSB**

Displays the least significant bit of a vector pin. This field is non-editable.

#### **Type**

Displays the pin type. You can modify the pin type through this column. By default, the pin type that was selected when the pin was created through the *Add New Pins* section is shown.

### ***Location***

Displays the location of the pin on the symbol. You can modify the location of a pin on the symbol from through this column.

### ***Input Load***

This column has two fields, *Low* and *High*. The values of *Low* and *High* fields determine the low and high input load values for the pin in mA. You can modify the low and high input load values for the pin through these fields.

### ***Output Load***

This column has two fields, Low and High. The values of output Low and High fields determine the low and high output load values for the pin in mA. You can modify the low and high output load values for the pin through these fields.

### ***Check Load***

Determines the value of the NO\_LOAD\_CHECK property for a particular pin type in the chips.prt file. This property is used by Rules Checker to execute the loading\_check Rules Checker rule. You can modify the value through this field.

### ***Check IO***

Determines the value of the NO\_IO\_CHECK property for a particular pin type in the chips.prt file. This property is used by Rules Checker to execute the inputio\_check Rules Checker rule. You can modify the value through this field.

### ***Check Direction***

Determines if the pin is to have a direction check. You can modify the status of the direction check through this field.

### ***Check Assert***

Determines whether assertion check will be done on the pin. You can modify the status of the check through this field.

### ***Check Output***

Determines whether an output check will be done on the pin. You can modify the status of the check through this field.

### **Unknown Loading**

Determines whether load checking will be done on the pin. You can modify the status of the check through this field.

### ***Pin Presence***

Displays the symbols and packages on which the pin is present.

### ***Shape***

Displays the name of the shape that is to be attached to the pin.

### ***Diff\_Pos***

Displays the differential pair name if the pin is the positive pin of a differential pair

### ***Diff\_Neg***

Displays the differential pair name if the pin is the negative pin of a differential pair

## **Rename**

This dialog box appears when you choose the *Rename* option from the pop-up menu displayed by right-clicking on a shape or symbol name in the shape or symbol tree. The new name you specify must be unique and should not contain any invalid character. The list of valid characters is as follows:

- A-Z
- 0-9
- -

## Extracted Shape Name

This dialog box appears when you extract a shape from a symbol with the *Extract Shape* option. The new name you specify must be unique and should not contain any invalid character. The list of valid characters is as follows:

- A-Z
- 0-9
- -

## Rename Pin

The Rename Pin dialog box appears when you choose the *Pins – Global Rename* option.

**Note:** Renaming a package/symbol pin results in the renaming of the pin across all the symbols and packages of the part.

The Rename Pin dialog box has the following elements:

### Old Name

Displays the existing pin names. The existing pin names are non-editable.

### New Name

Enables you to enter the new names for the pins. By default, the *New Name* column displays the existing pin names for each pin. However, the *New Name* column is editable and you can modify the pin names as required.

## Delete Package Pin

The Delete Package Pin dialog box appears when you choose the *Pins – Global Delete* option from the Package Editor.

**Note:** Deleting a pin through this dialog box pin results in the deletion of the pin across all the packages and symbols of the part.

The Delete Package Pin dialog box has the following elements:

### **Name**

Displays the logical pins that exist in a package.

### **Select**

Displays a check box next to each of the pins. To mark a pin for deletion, you need to select the check box next to the pin and click *OK*. On deleting a pin, the corresponding physical pin gets unmapped in the *Physical Pins* grid on the Package Pin page.

## **Add Properties**

The Add Properties dialog box appears when you choose the *Properties – Add* option. The Add Properties dialog box enables you to add properties to package or symbol pins. It has the following elements:

### **Name**

Enables you to enter a property. By default, the property is added to all the pins in the package.

### **Value**

Enables you to specify the value for the properties that you add in the *Name* column. By default, the value is added to all the pins in the package. In case you need to modify the value for a particular pin, you need to do so through the *Logical Pins* grid on the Package Pin page.

**Note:** By default, the `PIN_GROUP` property is available through the *Name* drop-down list. This property is used to determine the pins that are swappable with each other. For example, suppose a part has pins A, B, C, and D. If pins A and B can be swapped with each other and pins C and D can be swapped with each other, then the value of the `PIN_GROUP` property for pin A and B will be 1 and the value of the `PIN_GROUP` property for pins C and D will be 2.

## **Rename Package Pin Property**

The Rename Package Pin Property dialog box appears when you choose the *Properties – Rename* option. The Rename Package Pin Property dialog box enables you to rename a package pin property. It has the following elements:

### **Old Name**

Displays the existing package pin properties. The existing package pin properties are non-editable.

### **New Name**

Enables you to modify the package pin properties. By default, the *New Name* column displays the existing package pin properties. However, the *New Name* column is editable and you can modify the package pin properties as required.

## **Delete Package Pin Property**

The Delete Package Pin Property dialog box appears when you choose the *Properties – Delete* option. The Delete Package Pin Property dialog box enables you to delete one or more package pin properties. It has the following elements:

### **Name**

Displays the package pin properties.

### **Select**

Enables you to mark a property for deletion. To mark a property for deletion, you need to select the check box next to the property and click *OK*.

## **Add Physical Pin Numbers**

The Add Physical Pin Numbers dialog box appears when you choose *Footprint – Add Physical Pins Manually* from the Package Pin page of the Package Editor. The Add Physical Pin Numbers dialog box enables you to add physical pins for the package. It has the following elements:

### **Linear**

Enables you to add physical pins in a linear fashion, such as 1, 2, 3 or A1, A2 and so on.

## Grid

Enables you to enter the physical pins in a gridlike manner, such as AA, AB, AC and so on. You need to specify row labels and column labels. For example, to create physical pins such as AX, AY, AZ...ZX, ZY, ZZ, enter the row labels as A-Z and the column labels as X-Z.

## Delete Symbol Pin

The Delete Symbol Pin dialog box appears when you choose the *Pins – Global Delete* option from the Symbol Editor.

**Note:** Deleting a pin through this dialog box results in the deletion of the pin across all the views of the part.

It has the following elements:

### Name

Displays the pins that exist in a symbol.

### Select

Displays a check box next to each of the pins. To mark a pin for deletion, select the check box next to the pin and click *OK*.

## Symbol Pin Attributes

The Symbol Pin Attributes dialog box appears when you choose the *Pins – Attributes* option. This dialog box displays the existing attributes for all the symbol pins and enables you to modify them as required. It has the following elements:

### Name

Displays the names of the pins on the symbol. This field is non-editable.

### Shape

Displays the shapes for the symbol pins. You can modify the shape of any of the symbol pins. The possible shapes that a symbol pin can have is zero, line, dot, line-dot, line-clock, dot-

clock, line-dot-clock, and bubble. Line is the default pin shape for high-asserted pins. For low-asserted pins, the pin shape is taken from Setup.

### **Stub Length**

The Stub Length column displays the length of the symbol pin in grid units from the symbol outline. By default, the stub length is taken from Setup. You can modify the stub length of any of the symbol pins.

**Note:** When the stub length is changed, the coordinates of the connection point are not changed. This results in the changing of the stub size in the direction opposite to the connection point. To move the connection point, use the Move Pins option on the Symbol Pins page of the Symbol Editor.

### **Pin Name - Text Height**

Displays the height of the pin names. The default value for the text height is taken from Setup. You can modify the text height as required.

### **Pin Name - Alignment**

Displays the alignment of the pin names. You can modify the pin name alignment as required. The possible values are left, right, and center.

### **Pin Name - Rotation**

Displays the angle at which pin names are shown in the Symbol Viewer. You can modify the rotation values. The possible angles at which the pin names can be displayed are 0, 90, 180, and 270 degrees.

### **Pin Name - X and Y**

Displays the distance of the pin name with respect to the origin. The origin is taken as the 0 point of the x-y axis. You can modify these values as required.

### **Bubbled**

Displays whether the pin is bubbled. This field is enabled only if the pin shape is bubble. Selecting this option ensures that Design Entry HDL treats this pin as a bubbled pin. For more

information, see *Cadence Digital Library Standards* in *Design Entry HDL Libraries Reference*.

#### **Filled**

Displays the fill status of the connection hotspots. You can modify the fill status as required.

#### **Custom Shapes**

Displays the names of the custom shapes attached to the symbol pin. A maximum of two custom shapes can be attached to a pin.

## **Map Pin Name and Text**

The Map Pin Name and Text dialog box enables you to map the texts present on the symbol as pin text. By default, all text on a symbol that has not been associated with a pin will appear as unassociated pin text. These will include symbol text also. The Map Pin Name and Text dialog box has the following elements:

#### **Pin Name**

Displays the pin names as they appear in the *Logical Pins* list. This field is non-editable.

#### **Pin Text**

Shows the pin text that is associated with the pin name. If during part creation, the *Use Pin Names for Pin Text* check box was selected in the Setup Options, you will see the pin names appearing as pin texts in this column. You can unmap the pin texts by clicking the *Unmap* button. This field is non-editable.

#### **Unassociated Text**

Displays all texts on a symbol that are not associated with the pins.

#### **Associate**

Maps a selected text in the *Unassociated* text column to the selected pin name.

### **Unassociate**

Unmaps a pin text from a pin name. The pin text that is unassociated appears in the *Unassociated Text* column.

## **Rename Symbol Pin Property**

The Rename Symbol Pin Property dialog box appears when you choose the *Properties – Rename* option. The Rename Symbol Pin Property dialog box displays all the properties present on the symbol pins and provides the ability to rename one or more of the properties.

It has the following elements:

### **Old Name**

Displays the existing symbol pin properties. This field is non-editable.

### **New Name**

Enables you to modify the symbol pin properties. By default, for each property, the *New Name* value is same as the *Old Name* value. To modify a symbol pin property, change the entry in the *New Name* column.

## **Delete Symbol Pin Property**

The Delete Symbol Pin Property dialog box appears when you choose the *Properties – Delete* option. The Delete Symbol Pin Property dialog box enables you to delete one or more symbol pin properties.

**Note:** Deleting a symbol pin property results in the deletion of the property from all pins of the symbol.

It has the following elements:

### **Name**

Displays the symbol pin properties.

## Select

Enables you to mark a property for deletion. Selecting the check box and clicking *OK* results in the property getting deleted from all pins of the symbol.

## Move Pin

The Move Pin dialog box appears when you click the *Move* button in the Move Pins group box of the Symbol Pins page. *Move Pin* lets you move symbol pins in up, down, left, and right directions by one or more grids.

**Note:** When you move a pin, the connection hotspots are moved along with the pin stubs. Therefore, when moving a pin away from the symbol outline, ensure that the stub size is increased so as to maintain its connectivity with the symbol outline.

The Move Pin dialog box has the following elements:

### Direction

Determines the direction of the pin movement. The possible directions are up, down, left, and right.

### Grid units to move

Determines the number of grids by which the pins are to be moved in the specified direction.

## Select Package

The Select Package dialog box appears when you create a new VHDL or Verilog map file. This dialog box enables you to determine packages from which to derive the logical pin list for creating map files.

It has the following elements:

### Package Name

Displays all the packages for the part. The aliases also appear as separate entries in the package name list.

## Select

Select the packages from which the slot information will be taken to create the map files. The packages are used to derive the logical pin list and the slot information. Multiple packages can be selected only if the following is true:

- The packages have the same logical pin list.
- The number of function groups is same in all the packages.
- The logical pin list across the functions is same.
- The number of slots across the function groups is same for all packages.

## Modify Package

The Modify Package dialog box appears when you modify the pin list information for the map files. This dialog box enables you to determine packages from which to derive the logical pin list for creating map files.

It has the following elements:

### Package Name

Displays all the packages for the part. The aliases also appear as separate entries in the package name list.

## Select

Select the packages from which the slot information will be taken to create the map files. The packages are used to derive the logical pin list and the slot information. Multiple packages can be selected only if the following is true:

- The packages have the same logical pin list.
- The number of function groups is same in all the packages.
- The logical pin list across the functions is same.
- The number of slots across the function groups is same for all packages.

## Select Model

The Select Model dialog box enables you to select the VHDL/Verilog model file for creating Verilog/VHDL wrappers or map files. It has the following elements:

### Model Path

Specifies the path to the VHDL/Verilog model file.

### Browse Button

Selects either the *File* method or the *Lib:Cell:View* method of accessing a VHDL/Verilog model. For the *Lib:Cell:View* method, you must have an entry in the *cds.lib* file.

## Modify Model

The Modify Model dialog box enables you to model the VHDL/Verilog model file for existing Verilog/VHDL wrappers or map files. It has the following elements:

### Model Path

Specifies the path to the VHDL/Verilog model file.

### Browse Button

Selects either the *File* method or the *Lib:Cell:View* method of accessing a VHDL/Verilog model. For the *Lib:Cell:View* method, you must have an entry in the *cds.lib* file.

## Lib:Cell:View – Select Model

The Select Model dialog box appears when you select the *Lib:Cell:View* option in the Select Model dialog box. It enables you to access the VHDL/Verilog model files by the *Lib:Cell:View* method.

It has the following elements:

## **Library**

Displays the list of libraries available for the project. Select the library in which the model file is stored.

## **Cell**

Displays the list of cells available in the selected library. Select the cell in which the model file is stored.

## **View**

Displays the list of views for the selected cell. Select the view in which the model file is stored.

## **Import and Export Wizard**

The Import and Export Wizard page appears when you choose the *Import and Export* option from the *File* menu. This page displays the list of supported data format from which you can import and export part information.

## **Select Source**

The Select Source page appears when you select an import format in the Import Export wizard. This dialog box enables you to select the file that contains data in the format selected on the Import and Export Wizard page.

## **Select Destination**

The Select Destination page appears after you select the source file from which to import the data. This dialog box enables you to select the location where you want to store the imported data.

## **Select Capture Part**

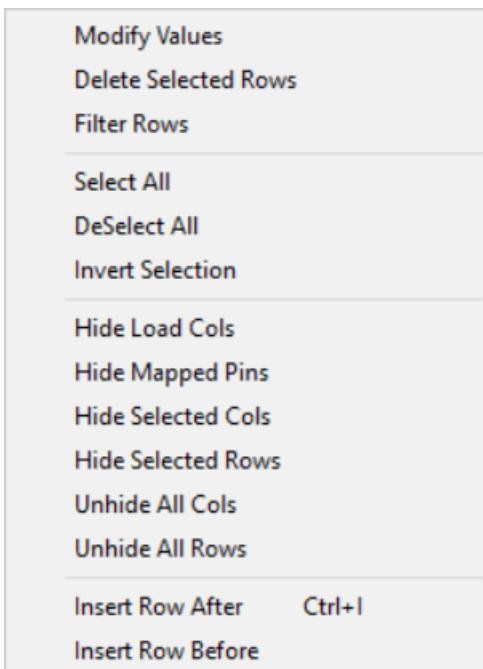
The Select Capture Part page appears after you select the Capture library from which to import the data. This dialog box enables you to select the Capture part to be imported.

Additionally, you can decide to import only the master component, create a primitive from each alias, or create a Design Entry HDL part for each alias.

## Preview of Import Data

The Preview of Import Data page appears after selecting the destination part name and library. This page displays how the part will be created from the source file. If required, you can make changes on this page. This enables you to modify the part before its actual creation.

When you right-click on the pin information displayed on this page, the following menu is displayed:



---

Field	Description
Modify Values	Use this option to change the value of a selected cell.
Delete Selected Rows	Use this option to delete one or more rows of pin information. To cancel the operation, you need to go back to the previous step by clicking the <i>Back</i> button on the Preview of Import Data page and re-import the data.

## Part Developer User Guide

### Dialog Box Help

---

<b>Field</b>	<b>Description</b>
Filter Rows	Use this option to display the Filter Rows dialog box and filter rows of pin information based on a pin detail. For example, you can specify a certain pin type in the Filter Rows dialog box to filter information for pins of that type only.
Select All	Use this option to select all the rows in the Logical Pins or Global Pins area.
DeSelect All	Use this option to clear the selection when all the rows are selected in the Logical Pins or Global Pins area.
Invert Selection	Use this option to clear the selected rows and select the rows that were not selected previously.
Hide Load Cols	Use this option to hide load columns in the Logical Pins or Global Pins Grid.
Hide Mapped Pins	Use this option to hide all the mapped pins from the Grid.
Hide Selected Cols	Use this option to hide one or more columns selected in the Logical Pins grid or the Global Pins grid.
Hide Selected Rows	Use this option to hide one or more rows selected in the Logical Pins grid or the Global Pins grid.
Unhide All Cols	Use this option to display all hidden columns in the current grid.
Unhide All Rows	Use this option to display all hidden rows in the current grid.
Insert Row After	Use this option to insert a row after the current row.
Insert Row Before	Use this option to insert a row before the current row.

---

## Select Package and Symbol(s)

The Select Package and Symbol(s) page appears after selecting the source Design Entry HDL part that is to be exported as the Capture part. You can select the packages and symbols that are to be exported as Capture parts.

**Note:** You can select a maximum of two symbols for conversion. Only those symbols that can be packaged into the selected package are displayed. Design Entry HDL symbols that have the `HAS_FIXED_SIZE` property do not appear in the list of symbols because Capture does not support fixed-size symbols.

Selecting the *Use Pin Name to write Capture Port Name* check box ensures that the symbol pin names are used as pin names in Capture. If this check box is not selected, the value of the PIN\_TEXT property is used to represent pin names in Capture.

## Select Package

The Select Package page appears after selecting the source part in ViewLogic import and : EDA XML export. You need to select the package that is to be imported or exported.

## Select Package

The Select Package page appears after selecting the source part in ViewLogic import and : EDA XML export. You need to select the package that is to be imported or exported.

## Select Package

The Select Package page appears after selecting the source part in ViewLogic import and : EDA XML export. You need to select the package that is to be imported or exported.

## Select Associated Package(s) or Unassociated Symbol(s)

The Select Associated Package(s) or Unassociated Symbol(s) page appears after selecting the source Design Entry HDL part that is to be exported as a ViewLogic part. You need to select the packages to be exported and also determine the part type.

## ECO Messages

The ECO Messages page displays the differences between the existing part and the source against which ECO is being done. You can review the list and select the differences that you want to import.

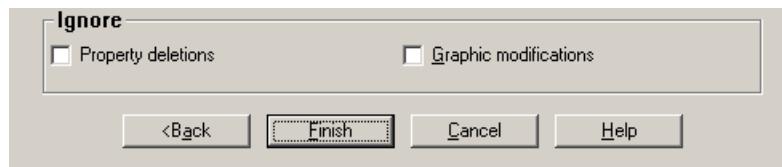
By default, the properties not found in the source file but existing in the destination are deleted. You can turn off this behavior by selecting the *Property deletions* check box in the *Ignore* section. Similarly, select the *Graphic modifications* check box in the *Ignore* section

## Part Developer User Guide

### Dialog Box Help

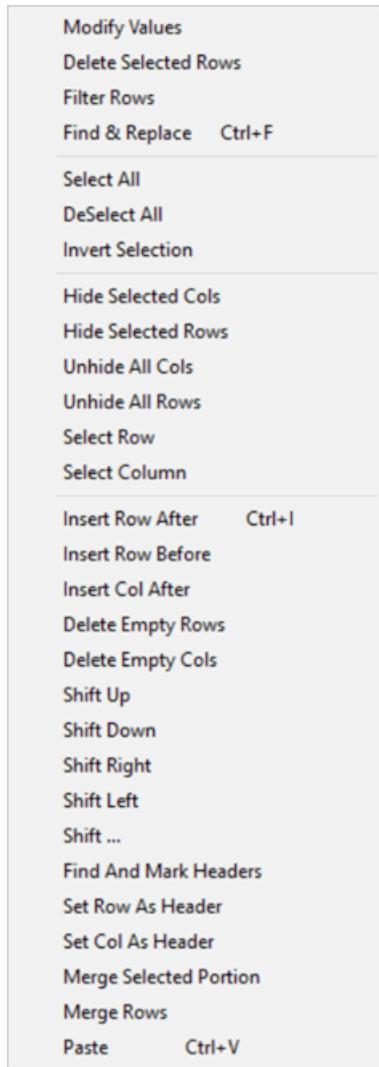
---

if you want to turn off the default behavior of retaining the symbol graphic modifications that have been done during ECO.



## Paste the data

The Paste the data page appears in Import Pin Grid and Import Pin Table. After pasting the data that was copied to the Clipboard for creating Design Entry HDL parts, you can massage the data through the various options in the pop-up menu:



## Preview of Derived Data

This page gives a preview of the data that has been extracted from the source. If required, you can make modifications before actually creating the part.

## Duplicate Pin Resolver

The Duplicate Pin Resolver dialog box appears in the import process if the import data contains more than one pin with the same name. To resolve the name clashes, you first select the type of the pin and then convert the pin to vector, scalar, or global. For example, if the import data contains three pins with the name A and you convert the pins to vector, the resolved pin names will be A<1>, A<2>, and A<3>. If you convert the pins to scalar, the resolved pin names will be A1, A2, and A3. In case you select the *Global* option from the *Convert* drop-down list, the pin names will remain A but the pins will be moved to the Global Pins section.

## Select Rows

This page appears after you specify the destination of the file to be imported. On this page, you can specify the profile to load and the range of rows to be imported. You can also preview the contents of the file being imported and select or ignore rows for import.

The Select Rows page has the following elements:

Element	Description
Select any profile to load	Displays the name of the profile file to be used
Browse	Enables you to browse and select the profile file to be used
Start	Enables you to specify the beginning of the range of rows to be imported. You can specify the following:
import at line number	■ a line number from which a set of consecutive rows is selected for import
import at line(s) after line <filter condition>	■ a condition for importing all rows that appear in the text file after a specified number of lines from the first occurrence of a row that contains a specified string either at the beginning or within it

## Part Developer User Guide

### Dialog Box Help

---

Element	Description
End	Enables you to specify the end of the range of rows to be imported. You can specify the following: <ul style="list-style-type: none"><li>■ import at line number</li><li>■ import at line(s) before line &lt;filter condition&gt;</li></ul>
Filter rows beginning with	Specifies the strings that the wizard should use to filter the rows that begin with the specified strings  <b>Note:</b> You can specify a maximum of two strings separated by a space.
Data Preview	Displays the data selected for import based on the conditions specified using the options displayed on the Select Rows page.  You can select or ignore rows from the displayed data by using the options on the pop-up menu.

## Select Delimiter(s)

This page appears in the import process after you have specified the range of data you want to import. On this page, you can select or specify the delimiter that should be used to parse the data. After specifying the delimiter, you can view the parsed data in the *Data Preview* section of this page.

## Part Developer User Guide

### Dialog Box Help

---

The Select Delimiter(s) page has the following elements:

<b>Element</b>	<b>Description</b>
Delimiters	Specify the delimiter by using the check boxes in this area.
Comma	Select this check box to specify comma as the delimiter.
Space	Select this check box to specify space as the delimiter.
Semicolon	Select this check box to specify semicolon as the delimiter.
Tab	Select this check box to specify tab as the delimiter.
Other	Select this check box to specify any character other than comma, space, semicolon, and tab as the delimiter.
Treat consecutive delimiters as one	Select this check box to allow multiple occurrences of the delimiter to be treated as one occurrence.
Text Qualifier	Select this check box to import values that contain the characters you have specified as delimiters.

## Select Columns

This page appears in the import process after the data to be imported is parsed using the delimiter of your choice and the parsed data is displayed in rows and columns. You can now select the columns you want to import and specify appropriate column headers.

The Select Columns page has the following elements:

<b>Element</b>	<b>Description</b>
Set first row as header	Select this check box to define the first row of the selected import data as the header.
Set pin number as pin name	Select this check box to ensure successful import even if the file being imported does not have a pin_name column, which is a required column for text import.
Data Preview	Displays the data selected for import in tabular format.  Select an appropriate header for each column from the drop-down list box displayed in the column. If a column header is not selected, the column is ignored.

## Select Views

On this page, you can specify if you want Part Developer to generate a symbol for the part that is being created from imported data. You can also select a footprint from a list of footprints displayed using `PSMPATH` and specify if the import settings are to be saved in a profile file.

The Select Views page has the following elements:

Element	Description
Import Data	Displays the number of rows and columns selected for import
Rows to be imported	
Columns to be imported	
Generate	
Generate Chips	This check box is selected by default.
General Symbol	Select this check box to generate a symbol for the part that is being created in the import process.
Associate Footprint	Select this check box to associate a footprint with the part being created.
Save Profile As	Click this button to save the settings you specified at various stages of import in a profile file.
	For a list of settings that are saved in a profile file, see <a href="#">Profile Use Model</a> on page 269.
Data Preview	Displays the data selected for import in tabular format.

## Select Footprint

This page appears in the footprint import process. On this page, you can select the footprint you want to import from a list of footprints displayed using `PSMPATH`.

## Select Footprint

The Select Footprint dialog box appears when you select the *Footprint – Select Using Ptf & Extract From Footprint* or *Footprint – Select Using Ptf & Verify With Footprint* option on the Package Pin page of the Package Editor. It has the following elements:

### Location

Displays the location.

### Name

Displays the name.

## Browse Jedecl Type

The Browse Jedecl Type dialog box appears when you click the browse button next to the *Jedecl Type* field on the *General* page of the Package Editor. This dialog box enables you to select a footprint for the package. It has the following elements:

### Name

Displays the list of available Allegro footprints. The list is derived from the \*.dra files stored in the location pointed to by the PSMPATH environment variable. Select the required footprint from this list.

## Browse Alt Symbol

The Browse Alt Symbols dialog box appears when you click the browse button next to the *A/t Symbols* field on the General page of the Package Editor. This dialog box enables you to select alternate footprints for the package. It has the following elements:

### Footprint

Displays the list of available Allegro footprints. The list is derived from the \*.dra files stored in the location pointed to by the PSMPATH environment variable. Select one or more footprints as alternate symbols for the package from this list.

## Type

Determines the footprint to be used when a part is placed on top or bottom of the board. The possible values are Both, Top, and Bottom. Select Both when the footprint can be used irrespective of whether the package is to be placed at the top or the bottom of the pcb. Select Top when the footprint is to be used only when the package is placed at the top of the pcb. Select Bottom when the footprint is to be used only when the package is to be placed at the bottom of the pcb.

For example, for part 74ls00, you may choose to use the dip14\_3 footprint when the package is placed at the top of the pcb and flat14 when the package is used at the bottom of the pcb. In such a case, select the dip14\_3 and flat14 footprints and select the types as top and bottom, respectively.

## Save All

The Save All dialog box appears when you select the *Save All* option from the *File* menu. This dialog box enables you to save all the parts open in Part Developer. It has the following elements:

### Cell Name

Displays the names of parts that are open in Part Developer.

### Save

Select the parts that you want to save. By default, all the objects are selected.

## Shapes – Save All

The Save All dialog box enables you to save unsaved changes in shapes. If you want to close a shape without saving the unsaved changes, deselect the *Save* check box for that shape and click *OK*. Clicking *Cancel* closes the dialog box without saving unsaved changes.

### Shape Name

Displays the names of shapes that are open in Part Developer.

## Save

Select the shapes that you want to save. By default, all the objects are selected.

## Annotate Generics

The Annotate Generics dialog box appears when you click the *Annotate Generics* button on the General page of the VHDL Map/Wrapper file editor. This dialog box enables you to determine which generics to annotate on the symbols.

By default, the Annotate Generics dialog box shows the VHDL model's generics, as well as the generics present in other VHDL map files/wrappers of the part. The *Annotate* option is shown selected for those generics that have been annotated on one of the symbols of the part. The value of the annotated generics is also shown.

It has the following elements:

### Annotation

Displays whether a generic is annotated on the symbol. Add/remove the generics annotated to the symbol by selecting the check box next to each of the generics. If you select a generic, the *Value to Annotate* column will display the value of the generic. You can change the value by clicking *Select Annotation Value*.

### Name

Displays the name of the generics.

### Type

Displays the type of the generics.

### Value to Annotate

Displays the value of the selected generic.

### Select Annotation Value

Launches the Select Value to be Annotated dialog box through which you can specify a value to the generic that is to be annotated to the symbol.

## Annotate Parameters

The Annotate Parameters dialog box appears when you click the *Annotate Parameters* button on the General page of the Verilog Map/Wrapper file editor. Add or remove the parameters annotated to the symbol by selecting the check box next to each of the parameters.



In addition to the parameters present in the selected Verilog model, this dialog box also shows all the parameters that are present in all the Verilog map/wrapper files for the part.

If you select a parameter, the *Value to Annotate* column will display the value of the parameter. The value can be changed by clicking *Select Annotation Value*.

The dialog box has the following elements:

#### Annotate

Displays whether a parameter is annotated on the symbol. Add or remove the parameters annotated to the symbol by selecting the check box next to each of the parameters.

#### Name

Name of the parameters in the Verilog model.

#### Type

Displays the types of the parameters.

#### Value to Annotate

Displays the value of the parameter that is annotated on the symbol.

## Select Annotation Value

Launches the Select Value to be Annotated dialog box through which you can specify a value to the parameter that is to be annotated to the symbol.

## Select Value to be Annotated

The Select Value to be Annotated dialog box appears when the *Select Annotation Value* button is clicked in the Annotate Generics/ Annotate Parameters dialog box. This dialog box enables you to specify new value for the generic/parameter that is to be annotated to the symbols.

It has the following elements:

### Generic/Parameter

Displays the generic/parameter that is annotated on the symbol. This field is non-editable.

### Value

Specify the value to the generic/parameter annotated on the symbol. If a part has multiple wrappers and the generic exists in these wrappers with differing values, then these values appear in the list. You can either select one of these or enter a new value.

## Add Logical Part

The Add Logical Part dialog box appears when you right-click on the *Logical Parts* entry on the Package Editor's General page and select the *New* option. It has the following element:

### Logical Part Name

Enter the new logical part name for the package.

## Add Physical Part

The Add Physical Part dialog box appears when you right-click on the *Physical Parts* entry on the Package Editor's General page and select the *New* option. It has the following element:

## **Physical Part Name**

Specify the new physical part name (package) for the selected logical part. The entry gets appended to the primitive in the `chips.prt` file.

## **Rename Physical Part**

The Rename Physical Part dialog box appears when you right-click on the *Physical Parts* entry on the Package Editor's General page and select the *Rename* option. It has the following element:

### **Pack Type**

Specify the new physical part name (package) for the selected logical part. The entry gets appended to the primitive in the `chips.prt` file.

## **Symbol Pin Property Attributes**

The Symbol Pin Property Attributes dialog box appears when you select the *Pins – Pin Text Attributes* option on the Symbol Pins page. This dialog box enables you to determine the attribute of the PIN\_TEXT property of the symbol pins. It has the following elements:

### **Pin Name**

Displays the logical pins on which the property is present. This field is non-editable.

### **Visibility**

Determines the display characteristics of the property and its value. The possible values are Invisible, Name, Value, and Both. If the *Visibility* is set to Invisible, then both the property name and its value are hidden. Setting *Visibility* to Name results in only the property name being visible on the symbol. If the *Visibility* is set to Value, then only the property value is visible on the symbol. Setting the *Visibility* to Both results in both property name and its value being visible on the symbol.

### **Text Height**

Determines the height of the property in grids.

## Alignment

Determines the alignment of the symbol pin property. The possible values are left, right, and center with respect to the pin.

## Rotation

Determines the angle at which the symbol pin property is displayed. The possible values are 0, 90, 180, and 270.

## Color

Determines the color in which the symbol pin property is displayed. The possible values are:

## X and Y

Determines the position of the symbol pin property from the origin.

## Save As

The Save As dialog box appears when you choose the *Save As* option from the *File* menu. It enables you to save the part by another name and/or in another library. It has the following elements:

### Library

Displays the list of libraries in the project. The list is created from the entries in the `cds.lib` file. You need to select a library in which to save the part.

### Cell

Specifies the name of the part by which you want to save the part in the selected library.

## Modify Properties

The Modify Properties dialog box appears when you right-click on a Verilog/VHDL wrapper and select the *Modify Properties* option. This dialog box enables you to modify the properties associated with a Verilog/VHDL wrapper file. It has the following elements:

## **Default Model**

Displays the model that is currently associated with the wrapper. You can change the model through the *Select Model* field.

## **UPPER\_CASE Property**

Displays the value of the `UPPER_CASE` property. You can change the value through the *Select Value* field.

## **Generate Symbol(s)**

The Generate Symbol(s) dialog box appears when you right-click on a package in the Cell Editor and select the *Generate Symbol(s)* option. This option enables you to generate a symbol directly from the selected package. You can also modify existing symbols to match its pin list with the selected package.

It has the following elements:

### **Select All**

Selects all function groups in the package. One symbol will be created for each function group. For example, if a part has two function groups, then two symbols will be created.

### **Deselect All**

Deselects all functions groups. No symbols will be created if you deselect all the function groups.

### **Function Group**

Displays all the function groups of the selected package.

### **Select**

Enables you to mark the function groups for which the symbols are to be created. One symbol is created for each selected function group.

## Add New or Modify

Enables you to determine whether to create a new symbol from the selected function groups or to modify an existing symbol. If you select an existing symbol, the pin list of the symbol is updated to match that of the selected function group.

## Modify Pin

The Modify Pin dialog box appears when you choose the *Pins – Global Modify* option from the Symbol Editor or the Package Editor. This dialog box enables you to modify the logical pin list of the part.

**Note:** Modifying a logical pin results in the pin getting modified across all the packages and symbols in which the pin is present.

It has the following elements:

### **Name**

Displays the pin names. This field is non-editable.

### **LSB**

Displays the least significant bit of a vector pin. This field is non-editable.

### **MSB**

Displays the most significant bit of a vector pin. This field is non-editable.

### **Type**

Displays the pin type. You can modify the pin type through this column. By default, the pin type that was selected when the pin was created through the *Add New Pins* section is shown.

### **Location**

Displays the location of the pin on the symbol. You can modify the location of a pin on the symbol through this column.

### ***Input Load***

This column has two fields, Low and High. The values of Input Low and High fields determine the low and high input load values for the pin in mA. You can modify the low and high input load values for the pin through these fields.

### ***Output Load***

This column has two fields, Low and High. The values of output Low and High fields determine the low and high output load values for the pin in mA. You can modify the low and high output load values for the pin through these fields.

### ***Check Load***

Determines the value of the NO\_LOAD\_CHECK property for a particular pin type in the chips.prt file. This property is used by Rules Checker to execute the loading\_check Rules Checker rule. You can modify the value through this field.

### ***Check IO***

Determines the value of the NO\_IO\_CHECK property for a particular pin type in the chips.prt file. This property is used by Rules Checker to execute the inputio\_check Rules Checker rule. You can modify the value through this field.

### ***Check Direction***

Determines if the pin is to have a direction check. You can modify the status of the direction check through this field.

### ***Check Output***

Determines whether an output check will be done on the pin. You can modify the status of the check through this field.

### ***Check Assert***

Determines whether assertion check will be done on the pin. You can modify the status of the check through this field.

## **Unknown Loading**

Determines whether load checking will be done on the pin. You can modify the status of the check through this field.

### ***Pin Presence***

Displays the symbols and packages on which the pin is present.

### ***Shape***

Displays the shape attached to the pin.

## **Filter Rows**

The Filter Rows dialog box enables you to view only those rows that match the filter values. For example, if you specify Analog pin type as the filter value, then only analog pins will be made visible.

## **Edit Global Mapping**

The Edit Global Mapping dialog box appears when you select Global Pin Map – Edit on the Package Pin page of Package Editor. It has the following elements:

### **Number**

Displays all the physical pins that are either mapped to global pins or are still unmapped.

### **Name**

Displays the global pins that are mapped to the physical pins. The cells of this column will appear blank next to the unmapped physical pins. To do the mapping, you need to select the global pins from the drop-down list and click OK.

## Modify Column Values

The Modify Column Values dialog box appears when you right-click on a selection of one or more rows of data and select *Modify Values*.

Depending on the data of the rows, the dialog box enables you to modify the data for the selected field for all the rows. Depending on the context of the selection, one of the following can be done:

- Replace

Replaces the current value with the specified value for the selected cells. For example, if you launch the dialog box on a selection of multiple logical pins and then change the pin type to, say BIDIR, then the pin type will change to BIDIR for all the selected pins.

- Increment by

Increments the current value by a factor of the specified value. The incrementation happens in the following way:

`old_value + (rowid * new value)`

The rowid starts from 0, where 0 is the first selected row.

For example, consider a symbol with five pins at the bottom of the symbol. These pins are separated from each other by 2 grids, with the first pin at -4, -2, 0, 2, and 4. If you select the *Increment By* option and specify the value 2, this will result in the pins getting moved to -4, 0, 4, 8, and 12. This implies that the pins moved 4 grids apart.

**Note:** The order of selection is critical for effectively using this feature.

- Add

Adds the specified value to the current value.

## Select Package to Associate

The Select Package to Associate dialog box appears when you right-click on a symbol name and select the *Add FunctionGroup to Package* option.

This dialog box enables you to associate the function group to any package that is not associated with this symbol.

## Specify The Symbol Size

The Specify The Symbol Size dialog box appears when you click *Set Size* on the Symbol Pins page of the Symbol Editor.

This dialog box enables you to specify the value of the SIZE property.

## Find & Replace

The Find & Replace dialog box appears when you press `Ctrl + F` in a grid or choose *Edit – Find & Replace* from the menu bar.

This dialog box enables you to find and replace the contents of a grid or columns. It has the following elements:

### Find What

Field to specify the search string.

### Replace With

Field to specify the replacement string.

### Options

Turns the additional search options on/off.

### Search In

Determines how the search will be performed. It can be either within the selected cells or the entire grid.

### Match Case

Determines the case-sensitivity of the search.

## **Search By**

Determines the order of the search, either by rows or columns.

## **Match Entire Cell Contents**

Controls the exact matching behavior. If this option is checked, the given search string is matched with the entire cell contents.

# **SI Model Interface Comparison**

The SI Model Interface Comparison dialog box appears when you select the *SI Model Interface Comparison* option from the Tools menu.

This enables you to compare DML models, IBIS models, Allegro footprints, and schematic parts.

## **Select Type**

Select the type of model.

## **Choose Model**

Browse and select the file that contains the selected model. For example, if you selected Schematic Part (Chips) in the Select Type field, you need to select a library part. The list of available packages or devices is displayed below the Choose Model list box. You need to select one of the devices/packages from the list.

## **Compare**

Compares the two selected models and displays the report.

# **SI Model Interface Comparison Results**

The SI Model Interface Comparison Results page appears when you compare DML models, IBIS models, Allegro footprints, and schematic parts.

### **Show All**

Displays the complete pin information for both models.

### **Show Common**

Displays the pins that are common across the two models.

### **Show Mismatched**

Displays the pins that are different across the two models. The mismatched pins can be further grouped as following:

- Same Pin Name

The pins with the same names across the two models are highlighted.

- Same Pin Number

The pins with the same pin numbers across the two models are highlighted.

- Same Pin Type

Pins with the same pin type across the two models are highlighted. This option is enabled only if you select the Same Pin Name or Same Pin Number option.

- Same I/O Buffer

Pins with the same pin buffers across the two models are highlighted. This option is enabled only if you select the Same Pin Name or Same Pin Number option.

**Note:** You can select a combination of these options. For example, by selecting Same Pin Name and Same Pin Number options, you can see those pins that have the same name and numbers across the two models.

### **Save As**

Saves the results in a text file.

**Note:** The sorting done on Name, Number and so on for Show All and Show Common section is not saved in the report file.

## New Shape

### Custom

#### Pin

This dialog box appears when you choose *File – New – Shape – CustomShape* or *File – New – Shape – PinShape* to create a custom shape or a pin shape. It has the following elements:

#### Shape

Enables you to specify a name for the new shape.

#### Path

Enables you to specify the location in which the shape is to be saved. By default, it displays the path specified in the *Default Shape Save Path* field specified in Setup.

## Open Shape

### Custom

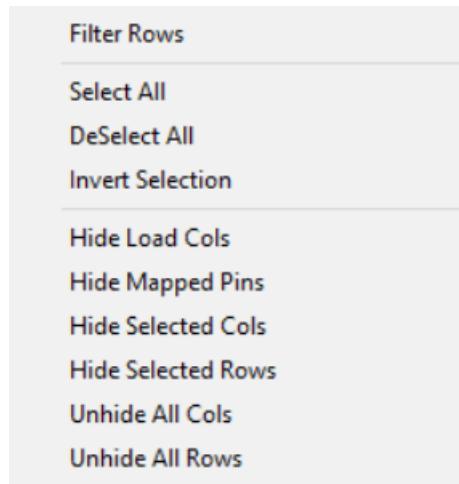
#### Pin

This dialog box appears when you choose *File – Open – Shape – CustomShape* or *File – Open – Shape – PinShape* to open an existing custom shape or pin shape.

### Row/Column Shown/Hidden Indicator

The *Row/Column Shown/Hidden Indicator* icon provides a visual indication of whether all the rows and columns are displayed in the grid.

If you right-click on the icon, the following pop-up menu is displayed:



## Name

The *Name* column displays the shape names.

## Path

The *Path* column displays the path of each custom or pin shape found at the location indicated by the *Shape Path* field in the Setup dialog box.

## Shape Viewer

The Shape Viewer displays the selected shape.

## Text Properties

The Text Properties dialog box enables you to modify the height, color, and justification of symbol or shape text.

## Template Application Wizard

The Template Application Wizard dialog box displays the symbol to which you want to apply a template and enables you to specify the template. You can also specify if the properties already existing on the symbol are to be deleted before applying the template.

**Note:** The symbol is displayed only for viewing and cannot be edited.

## Cadence Product Choices

Use this dialog box to choose a product suite from which you want to run Part Developer. All the product suites for which you have licenses and from which you can invoke Part Developer are listed in the Cadence Product Choices dialog box.

### How to Access

The Cadence Product Choices dialog box is invoked when:

- you are using the tool for the first time and on all subsequent invocations unless you specify the default choice.
- you choose *File – Change Product*.

### Setting a Default Product Choice

To prevent the Cadence Product Choices dialog box from appearing every time you run Part Developer, complete the following steps:

1. Select the product suite to be used as the default choice.
2. Select the *Use As Default* check box.

The Part Developer interface changes to reflect the selected product suite and will open with this product suite until you change the default setting.

3. Click *OK*.

The Part Developer title bar reflects the selected product suite.

Selecting the *Use as Default* check box writes the product choice in registry. As a result, for all future invocations, the tool will open with this product suite unless you change the default setting.

To change the default product suite, invoke the Cadence Product Choices dialog box from Part Developer by selecting *File – Change Product* and select the required product suite.

## **Specifying a Product Choice from the Command Line**

If you invoke Part Developer from the command line, you can use the `-product` option to prevent the Cadence Product Choices dialog box from appearing every time.

The syntax for using this option is

```
pdv -product license_string
```

The following `license_string` values invoke Part Developer with extended and core capabilities, respectively:

- `pcb_librarian_expert`
- `pe_librarian`

Therefore, the command for invoking Part Developer with the Allegro PCB Librarian XL license is:

```
pdv -product pcb_librarian_expert
```

## **Disabling License Checks**

To ensure that only the product suites for which you have licenses available are displayed in the Cadence Product Choices dialog box, the application checks with the license server for available licenses. The process of populating the dialog box with the list of available licenses takes some time.

However, if the time taken for displaying the Cadence Product Choices dialog box is high, you can use the `CDS_IGNORE_LIC_FEATURE` environment variable with its value set to `TRUE` to disable the procedure for checking for the available licenses. Using this variable ensures that the dialog box appears instantly but displays all the licenses using which you can launch Part Developer. From the list, you need to select the product suite for which you have the license available. For information on the available licenses, contact your license administrator.

## **Part Developer User Guide**

### Dialog Box Help

---

## **Part Developer User Guide**

### Dialog Box Help

---

---

## **Errors and Warnings**

---

Part Developer generates two categories of messages whenever it encounters a problem or a potential problem. These are:

■ **ERROR**

An ERROR message is displayed either in a situation where you cannot continue creating the part or switch between different Part Developer UIs until the error is fixed or when the part needs to be corrected before being used in the PCB design flow.

■ **WARNING**

A WARNING message is displayed when Part Developer encounters a possible error situation. You need to verify the warning and fix if required.

This chapter describes the ERROR and WARNING messages and their possible solutions. The following table describes the nomenclature followed in this chapter to identify an ERROR message and a WARNING message.

<b>Category</b>	<b>Nomenclature</b>
ERROR	E
WARNING	W

## Errors and Warnings

**SPLBPD-1, "When opening a project, you must specify a project name. Select the project from the Project drop-down list or use the Browse button to locate the .cpm file for the project.",E**

This error is generated if a project name is not specified in the Open Project dialog box. To work in Part Developer, you need to first open a project. If a project does not exist, you can create a new design or library project by choosing *File – New Project* in Part Developer.

**SPLBPD-2, "The part might not work in the flow because '(' is missing in the %s property value on line no. %s of the chips.prt file. Correct the syntax error manually in a text editor and reload the part.",E**

This error is generated when the starting ( is not found in a property value in the `chips.prt` file for `PIN_NUMBER`, `POWER_PINS`, or `NC_PINS`. For more information about the `chips.prt` file, see *Design Entry HDL Libraries Reference*.

**SPLBPD-3, "The part might not work in the flow because ')' is missing in the %s property value on line no. %s of the chips.prt file. Correct the syntax error manually in a text editor and reload the part.",E**

This error is generated when the ending ) is not found in a property value in the `chips.prt` file for `PIN_NUMBER`, `POWER_PINS`, or `NC_PINS`. For more information about the `chips.prt` file, see *Design Entry HDL Libraries Reference*.

**SPLBPD-4, "PIN\_NUMBER property value at line no. %s of the chips.prt file does not have a physical pin number. Map the logical pin name to a unique physical pin number and then save the part.",E**

This error is generated when the `PIN_NUMBER` property has no value in the `chips.prt` file. This can happen when a part has been saved without specifying a logical-to-physical pin mapping in a package. To know more about how to enter physical pins and mappings, see [Creating Packages](#) on page 121.

**SPLBPD-5, "Illegal pin range notation in PIN\_NUMBER property value on line no. %s of the chips.prt file for pin '%s'",E**

This error is generated when an incorrect pin range is specified as the value of the `PIN_NUMBER` property.

**SPLBPD-6, "The part might not work in the flow because the PIN\_NUMBER property value on line no. %s of the chips.prt file has a syntax error for pin '%s'. Correct the syntax error manually in a text editor and reload the part.",E**

This error is generated when the value for the PIN\_NUMBER property is incorrectly specified in the `chips.prt` file. For example, a pin number is entered as (`<AB`) instead of (`<AB>`). For more information about the `chips.prt` file, see *Design Entry HDL Libraries Reference*.

**SPLBPD-7, "The chips view cannot be loaded due to an invalid character found in the PIN\_NUMBER property value on line no. %s of the chips.prt file for pin '%s'. Make sure that the value is enclosed within valid characters and specified in the correct format.",E**

This error is generated in the following conditions:

- The PIN\_NUMBER property value in the `chips.prt` file contains `>` without any `<`.
- The PIN\_NUMBER property value is specified as `<n .n>` instead of `<n ..n>`, such as `<2.3>` instead of `<2..3>`.

For more information about the `chips.prt` file, see *Design Entry HDL Libraries Reference*.

**SPLBPD-10, "The cell cannot be created because you have not specified a cell name. Specify a unique and valid cell name in the Cell field and then click OK.",E**

This error is generated if you try to create a new part without specifying a cell name. Part Developer requires you to provide a valid cell name. For a list of valid characters, see [Supported Characters in Cell Names](#) on page 561.

**SPLBPD-11, "The cell cannot be created because the specified cell name has invalid character(s) %s. Rename the cell with valid characters only.",E**

This error is generated when an invalid character is specified in the cell name. The list of invalid characters is as follows:

- \
- :
- "
- '

- <

**SPLBPD-12, "Project path does not exist. Check if the CPM file exists at the specified location.",E**

This error is generated when the path to the project file is invalid. Check the path to the project file.

**SPLBPD-13, "The specified project path is incorrect. Make sure that the project path has a .cpm extension.",E**

This error is generated when the specified project file does not have a .cpm extension. Part Developer can only open project files that have the .cpm extension.

**SPLBPD-14, "Property %s cannot be added as a %s property because it is a reserved property. Reserved properties are predefined in propfile.prop, and their definition should not be altered.",E**

This error is generated when a reserved property, such as PART\_NAME, is specified as a property through one of the following methods:

- *Additional Properties* grid in the Package Editor
- *Properties – Add from Package Pin and Symbol Pins* pages
- Properties section in the Symbol Editor

The predefined list of reserved properties is available in the propfile.prop file located at the `<cds_inst_dir>\share\cdssetup\LMAN` location. The reserved properties are listed using the `NotAllowed` keyword. For example, the reserved properties for a package are listed as:

```
(NotAllowed  
"ALT_SYMBOLS,CLASS,JEDEC_TYPE,NC_PINS,PART_NAME,PHYS_DES_PREFIX,POWER_PINS,SWAP_INFO")
```

It is strongly recommended that you do not change the predefined list of reserved properties because they have special handling in Cadence flows and changing them may cause unpredictable behavior. To know more about the reserved properties and their use, see *Allegro Platform Properties Reference*.

## Part Developer User Guide

### Errors and Warnings

---

**SPLBPD-15, "The %s property value cannot be saved because the corresponding %s property name contains a null string. Specify the property name to ensure that the property value is saved.",W**

This warning is generated when a property value is specified without entering a property name. Part Developer requires you to specify a property for which you have entered a value. Part Developer does not save property values if corresponding property names are missing.

**SPLBPD-16, "The part might not work in the flow because of invalid character(s) %s found in the %s property name field. Rename the property with valid characters only.",E**

This error is generated when an invalid character is found in the property name. The invalid characters are listed below.

- !
- @
- #
- %
- ^
- &
- \*
- (
- )
- -
- +
- =
- ~
- `
- {
- }
- [

- ]
- \
- |
- :
- ;
- "
- '
- <
- >
- ,
- .
- ?
- /
- leading or trailing space

**SPLBPD-17, "Syntax error in the value field for the POWER\_GROUP property. Specify a new string with valid characters.",E**

This error is generated when an invalid value, such as an integer, is specified as the value for the POWER\_GROUP property. For more information about the POWER\_GROUP property, see *Allegro Platform Properties Reference*.

**SPLBPD-18, "The part might not work in the flow because of invalid character(s) %s found in the %s property value. Specify property values with valid characters only.",E**

This error is generated when invalid characters are found in the property value. The invalid characters are listed below.

- !
- "
- '

**SPLBPD-19, "Invalid RefDes Prefix value. Specify a value with not more than 30 characters.",E**

This error is generated when more than 30 characters are specified as the value for the RefDes prefix. For more details, see *Allegro/APD User Guide: Getting Started*.

**SPLBPD-20, "The cell cannot be saved because the cell name contains invalid character(s) %s. To rename the cell, use only the characters specified in the ValidCharSet list in propfile.prop.",E**

This error is generated when an attempt is made to save a cell that has invalid characters in its name. The list of valid characters is specified using the SaveAs – ValidCharSet directive in the `propfile.prop` file located at  
`<install_dir>\share\cdssetup\LMAN`.

**SPLBPD-23, "The cell cannot be saved because invalid character(s) %s is (are) found in the %s field. Specify the value with valid characters only.",E**

This error is generated when invalid characters are found.

For a list of valid values in different fields, see [List of Valid Values in Part Developer](#) on page 561.

**SPLBPD-24, "Pins with %s \* or \_N in the %s field cannot be added because these special characters are reserved for denoting low assertion. Rename the pins with valid characters only.",E**

This error is generated when \* or \_N is used as a pin name prefix. These two are reserved characters and used to denote low-asserted pins in the Cadence flow.

**SPLBPD-26, "Vector pin name '%s' does not have the termination character >. Add the > character at the end of the vector pin name (for example, A<3> and A<3..1>).",E**

This error is generated when a vector pin is specified without the terminating >, such as B<7, in the `chips.prt` file. Vector pins must be specified in the `pin_name<number of bits>` format, such as B<8>.

**SPLBPD-27, "Invalid character found in pin name '%s'. Make sure that pin names are specified using valid characters only.",E**

This error is generated when an invalid character is found in the pin name. See [Pin Naming](#) on page 564 for the list of characters that are valid for pin names.

**SPLBPD-28, "The cell cannot be saved because the %s property name %s is not unique. Delete the duplicate entry and then save the cell.",E**

This error is generated when the specified property name already exists. Property names could have been entered through the following methods:

- *Additional Properties* grid in the Package Editor
- *Package, Package Pin, Symbol, Symbol Pins*, and *PTF* options from *Tools – Setup*.

**SPLBPD-29, "The changed setup cannot be saved because you have not specified the pin grid size. Specify a positive non-zero real number in the Pin grid size field.",E**

This error is generated when the *Pin grid size* field on the *Tools – Setup – Symbol* page is left blank. The pin grid size must be a positive non-zero real number. For more information, see [Pin grid size](#) on page 93.

**SPLBPD-30, "The changed setup cannot be saved because the pin grid size is specified as 0. Specify a positive non-zero real number in the Pin grid size field.",E**

This error is generated when 0 is specified as the pin grid size. The pin grid size must be a positive non-zero integer. The pin grid size can be entered through the *Tools – Setup – Symbol* option.

**SPLBPD-31, "The changed setup cannot be saved because of invalid specification of the minimum symbol size. Make sure that the Height and Width fields have only non-zero, positive, and even integer values.",E**

This error is generated if the minimum grid height or minimum grid width on the *Tools – Setup – Symbol* page is specified as zero or a negative or odd integer or is left blank.

**SPLBPD-32, " The length of physical pin number %s is more than the PCB Editor default limit of 30 characters.",W**

This warning is generated when the number of characters in a pin number exceeds the default limit of 30 characters in Allegro PCB Editor. For more details, see *Allegro/APD User Guide: Getting Started*.

**SPLBPD-33, "The changed setup cannot be saved because the specified pin grid size is invalid. Specify a positive non-zero real number in the Pin grid size field.",E**

This error is generated when the value for the *Pin grid size* field is not a positive non-zero real number. The grid size can be entered through the *Tools – Setup – Symbol* option.

**SPLBPD-34, "The default value for low input load cannot be saved because the specified value is invalid. Specify a negative non-zero numeric value.",E**

This error is generated when the value specified for low input load is not a negative non-zero number. The default values for low input load can be specified through the *Tools – Setup – Package Pins* option. For more information, see Setting Up Package Pin Properties on page 87.

**SPLBPD-35, "The default value for high input load cannot be saved because the specified value is invalid. Specify a positive non-zero numeric value.",E**

This error is generated when the value specified for high input load field is not a positive non-zero number. The default values for high input load can be specified through the *Tools – Setup – Package Pins* option. For more information, see Setting Up Package Pin Properties on page 87.

**SPLBPD-36, "The default value for low output load cannot be saved because the specified value is invalid. Specify a positive non-zero numeric value.",E**

This error is generated when the value specified for low output load is not a positive non-zero number. The default values for low output load can be specified through the *Tools – Setup – Package Pins* option. For more information, see Setting Up Package Pin Properties on page 87.

**SPLBPD-37, "The default value for high output load cannot be saved because the specified value is invalid. Specify a negative non-zero numeric value.",E**

This error is generated when the value specified for high output load is not a negative non-zero number. The default values for high output load can be specified through the *Tools – Setup – Package Pins* option. For more information, see [Setting Up Package Pin Properties](#) on page 87.

**SPLBPD-38, "The length of primitive name %s is more than the maximum limit of %s characters. Rename the primitive or reconfigure the PrimitiveName\_MaxLength value in the cds.cpm file.",W**

This warning is generated when the number of characters in the primitive name exceeds the value configured in the `cds.cpm` file under the `PrimitiveName_MaxLength` directive. The `cds.cpm` file is located at `<install_dir>\share\cdssetup\projmgr`.

**SPLBPD-40, "The cell cannot be created because a cell with the specified name, %s, already exists in the selected library. Specify a unique cell name.",E**

This error is generated in the following conditions:

- The cell being created already exists in the selected library.
- The name of the cell to be created already exists in the library during FPGA import.
- The name of the cell to be created already exists in the library when importing part data from other data sources, except EDAXML.

**SPLBPD-41, "Invalid character(s) '%s' found in the PART\_NAME property value. Make sure that the PART\_NAME property value contains valid characters only.",E**

This error is generated when invalid characters are found in the `PART_NAME` property value. The list of invalid characters is as follows:

- ~
- `
- !
- \*
- (
- )

## Part Developer User Guide

### Errors and Warnings

---

- -
- +
- =
- |
- \
- }
- ]
- {
- [
- :
- ;
- "
- ,
- <
- >
- .
- ?

**SPLBPD-42, "Pin number(s) entered cannot be saved because they have invalid character(s) '%s'. Specify physical pin number(s) with valid characters only.",E**

This error is generated in the following conditions:

- Invalid character is specified as a physical pin number
- Invalid characters are specified in *Row Labels* and *Col Labels* fields in the Add Physical Pin Numbers dialog box
- Invalid characters are present in the *Linear* field in the Add Physical Pin Numbers dialog box
- Invalid characters are present in a pin number in the *Pin* column of the *Physical Pins* grid on the Package Pin page

## Part Developer User Guide

### Errors and Warnings

---

- Invalid characters or spaces are specified as pin numbers by editing the *Global Pins* grid or *Global Pin Map – Edit*
- Invalid characters or spaces are specified as pin numbers by editing the *Logical Pins* grid
- Invalid characters or spaces are specified in pin numbers by editing the *Physical Pins* grid

The invalid characters are as follows:

- \
- "
- !
- &
- @
- ~
- '
- ^
- <
- >
- .
- ,
- :
- ;
- {
- }

**SPLBPD-44, "The physical pin numbers cannot be extracted because the Linear field value is specified in an incorrect format. Make sure that the pin numbers are specified as a comma-separated list or a range indicated by the dash separator.",E**

This error is generated when the comma or dash separator is used incorrectly in the *Linear* field. The comma separator can be used to specify a list, such as 1, 2, 3, and the dash

separator can be used to specify a range, such as 1-100. For more information, see [Add Physical Pins Manually](#) on page 56.

**SPLBPD-45, "The property name cannot be saved because the specified name contains more than 2047 characters. Modify the property name so that the length of the new name is within the specified limit.",E**

This error is generated when the package property name or the package pin property name exceeds 2047 characters. Cadence flows do not support property names that exceed 2047 characters.

**SPLBPD-46, "The property value cannot be saved because the specified value contains more than 8192 characters. Reenter a value that is within the specified limit.",E**

This error is generated when the number of characters in the package property value or the package pin property value exceeds 8192 characters. Cadence flows do not support property values that exceed 8192 characters.

**SPLBPD-47, "The part cannot be used in the flow because one or more pins in the Global Pins grid are named NC. NC can be a valid pin name only for a pin of type NC. To correct the problem, change the pin type or the pin name.",E**

This error is generated when a non-NC global pin is named NC. Only pins of type NC can be named NC. For more information, see [Global Pins](#) on page 54 and [Adding Global Pins](#) on page 120.

**SPLBPD-48, "For a pin having pin type as NC in the Global Pins grid, only NC is a valid pin name. Any other name will be ignored.",W**

This warning is generated when an NC pin is given a name other than NC. This pin will not be saved as an NC pin. To make a pin NC, you must name the pin NC. Cadence flows do not support NC pin types in the body section of the `chips.prt` file if they are not named NC as well. For more information, see [Global Pins](#) on page 54 and [Adding Global Pins](#) on page 120.

**SPLBPD-49, "The part cannot be used in the flow because one or more pins in the Logical Pins grid are named NC. NC can be a valid pin name only for a pin of type NC. To correct the problem, change the pin type or the pin name.",E**

This error is generated in the following conditions:

- A logical pin is named NC but its type is not set to NC
- A logical pin in the CSV file being imported is named NC but its type is not set to NC

For more information, see [Global Pins](#) on page 54 and [Adding Global Pins](#) on page 120.

**SPLBPD-50, "The changed setup cannot be saved because the height specified as default is invalid. Choose Tools - Setup and specify only numeric or floating values.",E**

This error is generated when a non-numeric or floating value is specified as the default height for the following:

- Symbol text
- Symbol property
- Pin text
- Symbol pin property

You can set the default height values through the *Tools – Setup* option. For more information, see [Chapter 4, “Configuring Part Developer,”](#)

**SPLBPD-51, "Could not load view %s",E**

This error is generated when Part Developer fails to load a view, such as chips, symbol, entity, or PTF. This may happen due to some error in the files.

**SPLBPD-52, “The changed setup cannot be saved because the Minimum number of primitives to disable validation field contains an invalid value. Specify a non-zero positive integer and then click OK.”, E**

This error is generated if a non-integer or negative integer value is specified as the minimum number of primitives to disable validation in *Tools – Setup*. For more information, see [Validation](#) on page 86.

**SPLBPD-56, “Space character found in pack type '%s' for package '%s', removing it”,W**

This error message is generated if there is a space in the primitive name in the `chips.prt` file. The space is removed when the cell is opened in Part Developer. For example, suppose there is a pack type (primitive) called LS00 DIP. On opening this part, the pack type will be renamed LS00DIP.

## Part Developer User Guide

### Errors and Warnings

---

#### **SPLBPD-57, "Invalid character(s) '%s' found in the primitive name '%s'. Rename the primitive with valid characters only.",E**

This error is generated when invalid characters are found in the primitive value of the chips.prt file. The list of invalid characters is as follows:

- ~
- `
- !
- \*
- (
- )
- -
- +
- =
- |
- \
- }
- ]
- {
- [
- :
- ;
- "
- ,
- <
- >
- .
- ?

## Part Developer User Guide

### Errors and Warnings

---

#### **SPLBPD-58, "The primitive name '%s' is duplicate. Renaming it to %s\_PDVDUP%s.",W**

This warning is generated if a duplicate primitive is found in the `chips.prt` file. Part Developer renames the duplicate primitive name after generating the warning.

#### **SPLBPD-59, "The %s name '%s' cannot be saved because it has more than %s characters. Modify the part name so that the length of the new name is within the specified limit.",E**

This error is generated in the following conditions:

- The length of the logical part name in Logical & Physical Parts on the General page of the Package Editor exceeds 2047 characters.
- The length of the physical part name in Logical & Physical Parts on the General page of the Package Editor exceeds 2047 characters.

#### **SPLBPD-64, "Pin name '%s' at line no. %s of the chips.prt file is not unique for that primitive section. This pin will be ignored.",W**

This warning is generated when there are duplicate pin name entries in a package. These duplicate pin names will be deleted after some modifications are done to the package and the part is saved.

#### **SPLBPD-65, "PIN\_NUMBER property does not exist for some pin name in the chips.prt file. Ignoring such pins.",W**

This warning is generated when the `PIN_NUMBER` property is not found for some pins in a package. These pins are deleted when the package is modified and the part is saved.

#### **SPLBPD-66, "The name %s cannot be saved because it contains invalid character(s), '%s'. Specify a name with valid characters only.",E**

This error is generated in the following conditions:

- Invalid characters are specified when creating a new logical part or renaming a logical part
- Invalid characters are specified when creating a new logical part or renaming a physical part (pack type)

The list of invalid characters is as follows:

## Part Developer User Guide

### Errors and Warnings

---

- ~
- `
- !
- \*
- (
- )
- \_
- +
- =
- |
- \
- }
- ]
- {
- [
- :
- ;
- "
- ,
- <
- >
- .
- ?

## Part Developer User Guide

### Errors and Warnings

---

**SPLBPD-67, "PART\_NAME property value is not consistent with the primitive name(s). This value will be ignored.",W**

This warning is generated if the primitive names do not start with the PART\_NAME property value. Part Developer rewrites the value to a valid value if any changes are made in the chips.prt file and the part is saved.

**SPLBPD-69, "Invalid physical pin number(s) %s will not be saved. Specify valid pin numbers and then save the part.",W**

This warning is generated when you try to save a part with duplicate physical pin numbers. Duplicate pin numbers can be added only by manually editing the chips.prt file.

**SPLBPD-72, "Expected '(' character in OUTPUT\_LOAD property value. Ignoring the problem.",W**

This warning is generated when ( is not found in the OUTPUT\_LOAD property value.

**SPLBPD-73, "Expected ')' character in OUTPUT\_LOAD property value. Ignoring the problem.",W**

This warning is generated when ) is not found in the OUTPUT\_LOAD property value.

**SPLBPD-74, "Expected '(' character in INPUT\_LOAD property value. Ignoring the problem.",W**

This warning is generated when ( is not found in the INPUT\_LOAD property value.

**SPLBPD-75, "Expected ')' character in INPUT\_LOAD property value. Ignoring the problem.",W**

This warning is generated when ) is not found in the INPUT\_LOAD property value.

**SPLBPD-76, "Invalid character(s) %s found in pin name '%s'. Rename the pin with valid characters only.",E**

This error is generated when an invalid character is found in the pin name. The list of invalid characters is as follows:

■ ~

## Part Developer User Guide

### Errors and Warnings

---

- `
- !
- (
- )
- -
- +
- =
- |
- \
- }
- ]
- {
- [
- :
- ;
- "
- ,
- <
- >
- .
- ?
- \*

**Note:** The character \* is considered invalid only when it appears in the middle of a pin name. You can use \* at the end of a pin name to indicate that the pin should be considered low asserted.

**SPLBPD-77, "The cell cannot be loaded because the number of sections for pin '%s' specified on line number %s of the chips.prt file does not match the total number of**

**sections in the cell. Open the chips.prt file in a text editor, correct the mismatch, and reload the cell.",E**

This error is generated when the number of slots for a particular logical pin is less than the number of total slots. The number of slots is determined by the first PIN\_NUMBER entry in the `chips.prt` file. For example, if the pin number entry appears as `PIN_NUMBER='(1,13)'`, Part Developer assumes that this package has two slots. For more information about slots, see *Allegro Design Entry HDL Libraries Reference*. For more information on how to create slots in Part Developer, see [Entering Package Information](#) on page 123.

**SPLBPD-78, "Physical pin number '%s' is mapped to more than one pin. Specify a unique physical pin number for each pin on the Package Pin page.",E**

This error is generated in the following conditions:

- If a pin number is already mapped to a logical pin and a duplicate pin number is used in the *Global Pins* grid.
- If a pin number is already mapped to a pin name and an attempt is made to map it again to a different logical pin in the *Logical Pins* grid.

**SPLBPD-79, "Character length of '%s' property value exceeds the limit of 8192. Specify a new value within the character limit.",E**

This error is generated when the property value in a package exceeds 8192 characters.

**SPLBPD-80, "The part might not work in the flow because the PIN\_NUMBER property value on line no. %s of the chips.prt file has a mismatch in the number of bits for pin '%s'. Correct the syntax error manually in a text editor and reload the part.",E**

This error is generated when the pin number for a vector bit is missing in the `chips.prt` file. For example, consider a vector pin `A<3..0>`. If the pin numbers for this vector pin are provided as `PIN_NUMBER='(1,2,3)'`, the error will be generated because the pin number for the fourth bit is missing in the `chips.prt` file.

**SPLBPD-86, "Error in opening project %s",E**

This error is generated when Part Developer gets an error while opening a project. This can happen due to an error in the `.cpm` file.

**SPLBPD-88, "Error in opening cdslib %s",E**

This error is generated when Part Developer fails to open the `cds.lib` file. The `cds.lib` file contains the path to the libraries. For more information, see *Allegro Design Entry HDL Libraries Reference*.

**SPLBPD-91, "The project cannot be created because you have not specified a project name. Specify a unique name for the project.",E**

This error is generated when an empty project name is specified.

**SPLBPD-92, "The part might not work in the flow because the pin range notation in pin name '%s' is incorrect. Correct the syntax error manually and retry.",E**

This error is generated if an illegal pin range notation is used in the `chips.prt` file. The following types of errors will be caught:

- ‘pin name’< such as ‘A’<0
- ‘pin name < n’ such as ‘A<5’
- ‘pin name<‘ such as ‘A<

This error is also generated when an illegal pin range notation is found when importing part data from other sources.

**SPLBPD-93, "Ignoring the invalid CLASS property value '%s' found on line number %s in the chips.prt file. Specify a valid value or configure the list of valid values specified using the PackageClass keyword in the propfile.prop file.",W**

This warning is generated when the value of the CLASS property in the `chips.prt` file is not valid. The valid CLASS property values are defined through the `PackageClass` keyword in the `propfile.prop` file located at `<cds_inst_dir>\share\cdssetup\LMAN`.

**SPLBPD-94, "BODY\_NAME property value does not match the cell name. The mismatch will be auto-corrected when the cell is saved.",E**

This error is generated when the value of the BODY\_NAME property is not the same as the cell name. On saving the part, Part Developer will update the value of the BODY\_NAME property to match the cell name.

**SPLBPD-97, "The part might not work in the flow because sections %s and %s are identical in pin lists and mapping information. Make sure that each section of the part has unique pin information.",E**

This error is generated when the physical pin mapping of a logical pin is identical for two slots as shown in the graphic:

Logical Pins					
	Name	Type	S1	S2	
1	A	INPUT	1	1	
2	B	INPUT	2	2	
3	Y	OUTPUT	3	3	

**SPLBPD-98, "Metadata cannot be generated because page %s of the schematic view is corrupt. Make sure that the schematic is corrected.",E**

This error is generated when metadata generation for a cell fails because of corrupt data in the schematic.

**SPLBPD-99, "The part cannot be used in the flow because the length of pin name %s is more than the maximum limit of %s characters. Rename the pin or reconfigure the PinName\_MaxLength value in the cds.cpm file.",E**

This warning is generated when the number of characters in a pin name exceeds the value specified in the `cds.cpm` file under the `PinName_MaxLength` directive. The `cds.cpm` file is located at `<install_dir>/share/cdssetup/projmgr`.

**SPLBPD-101, "Path %s not found in the CDS file search paths",E**

This error is generated when the path to message files is not found in the CDS file search path.

**SPLBPD-102, "The section information cannot be saved because section % does not contain any logical pin. Make sure that at least one logical pin in each section of the part is mapped to a physical number.",E**

This error is generated when a slot is mapped to - for all logical pins. A multislot part must have at least one physical pin mapped to each slot. The other slots can be marked as -.

**SPLBPD-103, "Logical pin '%s' does not have the same physical pin numbers for the split part.",E**

This error message is generated when a logical pin in a split part is mapped to two different physical pins and then a save or change in selection is done in the Cell Editor.

**SPLBPD-104, "The part cannot be saved in the %s library because you do not have the required permissions. To save the part, choose a library for which you have write permissions or change the permissions for this library. ",E**

This error is generated when there is no write permission for the given library location. You can either change the permission of the library or choose another library to save your cell.

**SPLBPD-105, "No write permissions in %s. Cannot overwrite.",E**

This error is generated when there is no write permission for the part in the library. You can either change the permissions of the library or choose another library to save your cell.

**SPLBPD-106, "Saving of cell %s in library %s is not successful",E**

This error is generated when Part Developer fails to save the part in a library. The error message number 105 will describe the reason for the failure of save.

**SPLBPD-107, "Cannot run executable van because file %s is not present. Make sure that the verilog.v file is present in the entity folder.", E**

This error is generated when the executable needed to run van is not found in the path.

**SPLBPD-108, "Pin %s '%s' is invalid because a pin %s cannot begin or end with :. Specify the value with valid characters only.",E**

This error is generated when a pin name begins or ends with a :.

**SPLBPD-109, "The pin cannot be added because the value specified in the LSB field is invalid. Specify a positive integer value that is smaller than the MSB value and then click Add.",E**

This error is generated if the specified LSB value is not a positive integer.

**SPLBPD-110, "The pin cannot be added because the value specified in the MSB field is invalid. Specify a positive integer value that is greater than the LSB value and then click Add.",E**

This error is generated when a value specified in the MSB field is not a positive integer.

**SPLBPD-113, "One or more renamed package pin properties in the modify grid are empty. Specify a new name for each property listed in the Old Name column and then click OK.",E**

This error is generated when a null value is entered as the new name for the package pin property being renamed. Part Developer cannot save property names with null values.

**SPLBPD-115, "Invalid syntax for the POWER\_PINS property on line no. %s at token: ':'. Correct the syntax error in chips.prt or add the power pin information in the Package Editor.",W**

This warning is generated when the value for the POWER\_PINS property is syntactically incorrect, such as more than one : is specified in the value. For example,

```
POWER_PINS='(GND::11)' ;
```

For more information about the POWER\_PINS property, see *Allegro Platform Properties Reference*.

**SPLBPD-116, "Unknown pin type %s. Changing it to UNSPEC.",I**

This warning is generated when the pin type for a pin is not supported by Part Developer. The pin types for such pins are changed to UNSPEC.

**SPLBPD-117, "Logical pin '%s' is not mapped to any physical pin. To map the pin, specify a physical pin number in the %s column of the Logical Pins grid.",E**

This error is generated when no physical pin is mapped to a slot of a logical pin. If a logical pin is not present in a certain slot, select the slot and click *Map To -*.

**SPLBPD-118, "The part cannot be used in the flow because one or more global pins are not mapped. Select each unmapped pin in the Global Pins grid and map it to physical pin numbers.",E**

This error is generated when a global pin exists and is not mapped to a physical pin. You must map all global pins to one or more physical pins. For more information, see [Global Pin Map](#) on page 54 and [Global Pin Map – Edit](#) on page 133.

**SPLBPD-120, "The entity view for cell %s cannot be created because the executable newgenasym cannot be run. Check the Path environment variable to ensure that the newgenasym executable is in your path. ",E**

This error is generated when Part Developer is unable to run the newgenasym executable, which is responsible for creating the entity view of the part. The newgenasym executable is stored at `<cds_inst_dir>\tools\bin`.

**SPLBPD-121, "Unable to create the entity view because the executable van cannot be run. Check the Path environment variable to ensure that the van executable is in your path.",E**

This error is generated when there is a problem in executing van. Check the path environment variable to ensure that the van executable is in your path. The van executable is stored at `<cds_inst_dir>\tools\fet\bin`.

**SPLBPD-122, "The pin name cannot be saved because the range is specified using letters. Specify a range of only numeric values within angular brackets (for example, A<1..5>).",E**

This error is generated when < or > is used to specify the range for non-numeric values. For example, you can specify a pin name as Z<1..5> in the *Logical Pins* grid to add 5 bits of a vector pin Z. However, trying to add a pin range such as <A..Z> will generate the error.

**SPLBPD-123, "Unable to run %s. Check your PATH settings.",E**

This error is generated when Part Developer fails to launch the selected tool. This can happen due to incorrect or incomplete PATH settings, such as when compiling VHDL or Verilog wrappers using the *Tools – Verify* option and the executables `ncvlog` or `ncvhdl` are not found to compile the wrappers.

## Part Developer User Guide

### Errors and Warnings

---

#### **SPLBPD-124, "Basenames of two pins cannot be same. Rename or delete one of the pins.",E**

This error is generated when two logical pins have the same base names. You must change one of the base names.

#### **SPLBPD-125, "The symbol cannot be saved because pins %s and %s overlap at %. To save the symbol, drag one of the overlapping pins and place it at different X, Y coordinates or assign a different Position value on the Symbol Pins page to one of the pins.",E**

This error is generated when the X and Y coordinates of two symbol pins are same as shown in the following graphic:

	Name	Text	PinType	Sized	Location	Position	PIN_DELAY
1	A<SIZE-1...	A	INPUT	<input checked="" type="checkbox"/>	Left	<input type="text" value="4"/>	
2	B<SIZE-1...	B	INPUT	<input checked="" type="checkbox"/>	Left	<input type="text" value="2"/>	
3	Y<SIZE-1...	Y	OUTPUT	<input checked="" type="checkbox"/>	Right	<input type="text" value="4"/>	

To eliminate the possibility of errors when using such a symbol, you need to move the overlapping pins to different coordinates or assign different *Position* values.

#### **SPLBPD-126, "No unique non-common pin found across sections %s and %s for the split part. Make sure that at least one unique pin is mapped to each section.",E**

This error is generated when there is no unique pin mapped to the slots of a split part. Cadence flow requires that for a split part, at least one unique pin should be mapped to each slot. For example, consider a split part with four slots S1, S2, S3, and S4 and four pins A,B,C, and D. Then, at least one pin, say A, should be mapped only to a slot S1. If it is mapped to any other slot, say S2, then this error will be generated.

#### **SPLBPD-127, "The part cannot be saved because logical pin(s) %s is (are) not mapped to any section. Map the pin to at least one section and then save the part.",E**

This error is generated when logical pins are mapped to - for all sections. You must map a logical pin to a physical pin for at least one section.

**SPLBPD-128, " The split part cannot be saved because section %s does not have any unique logical pin. Re-assign pins to ensure that each section of the split part has at least one unique logical pin.",E**

This error is generated when no unique non-common pin is found for a slot of a split part at the time of saving the cell.

**SPLBPD-129, "Invalid filename. Make sure that the model filename has a .v extension.",E**

This error is generated in the following conditions:

- The filename for a model does not have the .v extension when adding or modifying a model for a Verilog wrapper or map file.
- The name of a Verilog file selected for import does not have a .v extension.

**SPLBPD-130, "Invalid filename. Make sure that the model filename has a .vhdl extension.",E**

This error is generated in the following conditions:

- The filename for a model does not have the .vhdl extension when adding or modifying a model for a VHDL wrapper or map file.
- The name of a VHDL file selected for import does not have a .vhdl extension.

**SPLBPD-131, "Pin name cannot begin with ':'. Specify a name with valid characters only.",E**

This error is generated when a pin name begins with a colon.

**SPLBPD-132, "Invalid character '\*' found in the Pin Name field. If the pin is low-asserted, specify the reserved character '\*' as the suffix. Otherwise, rename the pin with valid characters.",E**

This error is generated when the \* character is found in the pin name. This happens because \* is a reserved character and used by Part Developer to determine pin assertion. The \* character can be used only in the pin name suffix. For more information about pin assertion, see [Setting Up Defaults for Low-Asserted Pins and Split Parts](#) on page 83.

**SPLBPD-133, "Pin name cannot end with ':'. Specify a name with valid characters only.",E**

This error is generated when a pin name ends with a colon.

**SPLBPD-134, "The pin cannot be added because the value(s) specified in To and From fields is (are) invalid. Specify either letters or numbers in both the fields (for example, 1 and 3 or A and C) and then click Add.",E**

This error is generated when a mix of letters and numerals is specified in the *To* and *From* fields in the Add Pin dialog box. You must specify either letters or positive integers in the *To* and *From* fields.

**SPLBPD-135, "Pins with PIN\_GROUP %s do not have the same pin type. Check PIN\_GROUP values and make sure that pins with the same PIN\_GROUP value also have the same pin type.",E**

This error is generated when pins with the same value for the PIN\_GROUP property have different pin types. For more information about the PIN\_GROUP property, see *Allegro Platform Properties Reference*.

**SPLBPD-136, "Character(s) %s are not allowed in the name field. Make sure that the name field contains valid characters only.",E**

This error is generated when an invalid character is specified while renaming a symbol or VHDL or verilog map file or model in Cell Editor tree. The list of invalid characters is as follows:

- :
- /
- \
- <
- >
- ?
- "
- |

**SPLBPD-138, "%s already exists.",E**

This error is generated when the specified file or view already exists. This can happen in the following conditions:

- The file to be created in Viewlogic export already exists.
- The file to be created in EDAXML export already exists.
- The file to be created in CSV export already exists.
- The name specified when renaming a symbol, a wrapper, or a model already exists.

**SPLBPD-139, "Cannot rename the model to %s because the name is reserved. Specify a name other than chips, entity, and part\_table.",E**

This error is generated if any of the VHDL or Verilog models or wrappers is renamed entity, chips, or part\_table.

**SPLBPD-140, "The symbol cannot be saved because pin(s) %s is (are) not on the standard grid. To make sure that all pins are on the standard grid, first select all the pins in the Symbol Pins pane and then click the up and down arrow buttons in the Move Pins area to move the pins up by 1 grid unit and then down by 1 grid unit.",E**

This error is generated when one or more symbol pins are not on the standard grid. In Part Developer, the standard grid is the grid with the default pin grid size, which is same as the default symbol grid size of Design Entry HDL.

The problem of symbol pins not on the standard grid can happen if the minimum height or width of a symbol is changed using *Tools – Setup – Symbol* such that pins go off the standard grid or a pin is assigned coordinates in `symbol.css` without taking into account the standard grid size.

To correct the problem:

1. Open the symbol in Part Developer.
2. In the *Symbol Pins* pane, select all the pins in the pin grid by using the Shift key and the left mouse button.
3. In the *Move Pins* area of the Symbol Pins pane, click the up arrow button and the down arrow button.
4. Save the part.

**SPLBPD-141, "The cell cannot be saved because symbol property SPLIT\_INST contains an invalid value. Make sure that the SPLIT\_INST value is always TRUE.",E**

This error is generated when an invalid value is found for the SPLIT\_INST property. The only value that this property can have is TRUE. For more information on the SPLIT\_INST property, see *Allegro Platform Properties Reference*.

**SPLBPD-142, "Pin text visibility cannot be set because symbol property PIN\_TEXT\_VISIBLE contains an invalid value. Make sure that the PIN\_TEXT\_VISIBLE value is either TRUE or FALSE.",E**

This error is generated if a value other than TRUE or FALSE is specified as the value for the PIN\_TEXT\_VISIBLE property.

**SPLBPD-146, "The part might not be usable in the flow because the pin types of pin(s) '%s' are not same in all of the packages. Make sure that this pin type specification is intended.",W**

This error message is generated when the pin type of a particular pin is different across different packages. Typically, the pin type of a particular pin should be same for each package. Examine carefully if the pin type should be different across packages.

**SPLBPD-147, "Pin types of all the bits of vector pin '%s' are not same. Make sure that this pin type specification is intended.", W**

This warning is generated when the pin type of the different bits of a vector pin are different. Examine carefully whether the pin type of the different bits of a vector pin should be different.

**SPLBPD-150, "The %s file cannot be opened. Check the permissions to ensure that read access is available.",E**

This error is generated when Part Developer is unable to open the chips.prt file or the map file in read mode. Check the permissions to ensure that read access is available.

**SPLBPD-151, "The chips file %s cannot be parsed because of an error on line no. %s and token: %s. Correct the syntax error manually and reload the chips file.",E**

This error is generated when Part Developer fails to read the chips.prt file due to syntactical errors. To know more about the syntax of the chips.prt file, see *Allegro Design Entry HDL Libraries Reference*.

**SPLBPD-152, "Map file %s cannot be parsed because of an error at line no. %s and token: %s. Correct the syntax error manually and reload the map file.",E**

This error is generated when Part Developer fails to read a map file because of errors in the map file. To know more about the syntax of map files, see *Allegro Design Entry HDL Libraries Reference*.

**SPLBPD-156, "The symbol cannot be modified because you have selected the same symbol for more than one function group. Select a unique symbol for each function group.",E**

This error is generated if the *Generate Symbol(s)* option is used for a package having more than one function group and the symbol selected for modification is the same for two or more of these functional groups.

**SPLBPD-157, "The specified low input load value cannot be saved because the value is invalid. Specify a negative non-zero numeric value.",E**

This error is generated when the value specified for the low input load field is not a negative non-zero number.

**SPLBPD-158, "The specified high input load value cannot be saved because the value is invalid. Specify a positive non-zero numeric value.",E**

This error is generated when the value specified for the high input load field is not a positive non-zero number.

**SPLBPD-159, "The specified low output load value cannot be saved because the value is invalid. Specify a positive non-zero numeric value.",E**

This error is generated when the value specified for the low output load field is not a positive non-zero number.

**SPLBPD-160, "The specified high output load value cannot be saved because the value is invalid. Specify a negative non-zero numeric value.",E**

This error is generated when the value specified for the high output load field is not a negative non-zero number.

**SPLBPD-162, "No pin is mapped in mapfile.package.model '%s.%s'. Map all pins and then save the file.",E**

This error is generated if all pins are left unmapped in VHDL or Verilog map files and an attempt is made to save the map file.

**SPLBPD-163, "Incomplete mapping done for pin(s) '%s' in mapfile.package.model '%s.%s'. Map the unmapped pins and then save the file.",E**

This error message is generated when a pin is not mapped to all the slots in a map file or some pins are left unmapped.

**SPLBPD-164, "Package %s and package(s) %s do not meet the conditions for clubbing. Refer Help for details.",E**

This error message is generated when multiple packages selected to determine the different information fails to meet one or more of the following conditions:

- The packages have the same logical pin list.
- The number of function groups is same in all the packages.
- The logical pin list across the functions is same.
- The number of slots across the function groups is same for all packages.

For more information, see [Chapter 9, “Working with VHDL Wrappers and Map Files”](#) and [Chapter 10, “Creating Verilog Wrappers and Map Files.”](#)

**SPLBPD-165, "The symbol cannot be renamed because the specified name is invalid. Reenter the name in the format sym\_n, where n is a non-zero positive integer.",E**

This error is generated when the name specified for a symbol is other than *sym\_n*. The PCB design flow requires that all symbols be named in the format *sym\_n*, where *n* is a non-zero positive integer. For more information, see *Allegro Design Entry HDL Libraries Reference*.

**SPLBPD-166, "Pin(s) '%s' in mapfile.package.model '%s.%s' are missing in the package. Add the pins in the package or remove them from the map file.",E**

This error message is generated when a pin is found in the map file but is not present in the corresponding package. For more information, see [Chapter 9, “Working with VHDL Wrappers and Map Files”](#) and [Chapter 10, “Creating Verilog Wrappers and Map Files.”](#)

**SPLBPD-167, "Package(s) %s in mapfile %s is (are) missing in the chips file. Add the package or delete the reference.", E**

This error message is generated when a map file has been created using more than one package and one of the packages is deleted. For more information, see [Chapter 9, “Working with VHDL Wrappers and Map Files”](#) and [Chapter 10, “Creating Verilog Wrappers and Map Files.”](#)

**SPLBPD-168, "The model cannot be pasted in package %s because the package is not in sync with package %s from which the model was copied. Use the Interface Comparison tool to synchronize the two packages. ",E**

This error message is generated when an attempt is made to copy a model from one package to another and the packages are not synchronized. You can synchronize the package by using the Interface Comparator feature of Part Developer. For more information, see [Chapter 14, “Interface Comparator.”](#)

**SPLBPD-169, "'Ignoring the GROUND\_NETS property because of invalid characters ( and ) in the specification. Open the chips.prt file in a text editor and specify the property in the GROUND\_NETS = <pin name>; syntax.",W**

This warning is generated when the value of the GROUND\_NETS property has ( and ) in it. The following is an example of GROUND\_NETS property specification in the `chips.prt` file:

```
GROUND_NETS='GND' ;
```

For more information on the GROUND\_NETS property, see [Allegro Platform Properties Reference](#).

**SPLBPD-170, "Pin(s) %s is (are) not present in any package or symbol. If the HAS\_FIXED\_SIZE value has been reduced, reload the part. Otherwise, you can choose Pins - Add from the Package Pin page and delete these pins.",W**

This warning is generated when a pin has been entered for a part but has not been added to any of the packages or symbols. You can see the pins in the Add Pin dialog box and delete them if required. Otherwise, you can add them to packages or symbols. For more information on how to access the Add Pin dialog box and to add pins to a package or a symbol, see [Adding Logical Pins on page 112](#).

**SPLBPD-171, "There are no VHDL wrappers to be compiled.",E**

This error is generated when the *VHDL Compilation* verification option is selected and there are no VHDL wrappers in the cell. For information on how to create VHDL wrappers, see [Chapter 9, “Working with VHDL Wrappers and Map Files.”](#)

**SPLBPD-172, "There are no Verilog wrappers to be compiled.",E**

This error is generated when the *Verilog Compilation* verification option is selected and there are no Verilog wrappers in the cell. For more information on how to create Verilog wrappers, see [Chapter 10, “Creating Verilog Wrappers and Map Files.”](#)

**SPLBPD-174, “Invalid PIN\_DELAY value. Specify a numeric value in <Pin\_Num:Prop\_Value;> format.”,E**

This error is generated if an alphanumeric value is specified or no units are specified for the *Pin Delay* option in *Tools – Setup – Package Pins*. The error is displayed in the following situations:

1. Cell selection is changed in the Logical Pins grid.
2. The cell is saved or opened.

For more information on `PIN_DELAY`, see *Allegro Platform Properties Reference*.

**SPLBPD-177, "The specified low input load value cannot be saved because the value is invalid. Specify a negative non-zero numeric value.",E**

This error is generated when the value specified for the low input load field is not a negative non-zero number.

**SPLBPD-178, "The specified high input load value cannot be saved because the value is invalid. Specify a positive non-zero numeric value.",E**

This error is generated when the value specified for the high input load field is not a positive non-zero number.

**SPLBPD-179, "The specified low output load value cannot be saved because the value is invalid. Specify a positive non-zero numeric value.",E**

This error is generated when the value specified for the low output load field is not a positive non-zero number.

**SPLBPD-180, "The specified high output load value cannot be saved because the value is invalid. Specify a negative non-zero numeric value.",E**

This error is generated when the value specified for the high output load field is not a negative non-zero number.

**SPLBPD-181, "Pin '%s' on line no. %s of the chips.prt file will be ignored because the primitive section contains another pin, %s, with the same basename and hyphen prefix. Make sure that there is no loss of information."E**

This error is generated if a pin name with a hyphen prefix already exists in `chips.prt` and another low-asserted pin with the same name is added by editing `chips.prt` and saved. You will get the error when the cell is reopened. For more information on how Part Developer handles low-asserted pins, see [Setting Up Defaults for Low-Asserted Pins and Split Parts](#) on page 83.

**SPLBPD-182, "Pin '%s' on line no. %s of the chips.prt file will be ignored because the primitive section contains another pin, %s, with the same basename but a different assertion level. Reenter the ignored pin information and ensure that no two pins with the same name and different assertion levels exist in the same package. ",E**

Part Developer does not allow pins with the same name but different assertion levels to exist in the same package. Depending on the value selected in Setup, pins with \* or \_N are treated as low-asserted pins. Therefore, this error message is generated when two pins exist, where one has \* or \_N as its suffix, such as A and A\_N. In such a case, pin A is ignored. For more information on how Part Developer handles low-asserted pins, see [Setting Up Defaults for Low-Asserted Pins and Split Parts](#) on page 83.

**SPLBPD-183, "Logical pin %s is mapped to more than one physical pin number in the same slot. To save a single valid mapping, save and reload the part.",E**

This error message is generated when a logical pin is mapped to different physical pins in the same slot.

**SPLBPD-184, “Ignoring logical pin %s because the pin is not mapped to any section. If this is not desired, open the chips.prt file in a text editor and specify the mapping information in the correct syntax.”, E**

This error message is generated if a cell has a symbol pin where the value for the PIN\_NUMBER property in the chips.prt file is empty for that particular pin name and the corresponding cell is opened in Part Developer. Part Developer will not show the pin name in the symbol and the package pin list. For more information on the PIN\_NUMBER property, see *Allegro Platform Properties Reference*.

**SPLBPD-185, “The CLASS property value for the package cannot be MECHANICAL because the package has logical pins. Redefine the CLASS property value by choosing a valid value from the Class drop-down list on the General page.”, E**

This error message is generated if in the General tab in Package Editor, the CLASS property is given the value MECHANICAL and then an attempt is made to change to some other panel through the Cell Editor.

**SPLBPD-186, “Physical pin %s is not mapped to any functional group for this logical pin. Specify the pin delay value in the proper row.”, E**

This error is generated if a logical pin in the *Logical Pins* grid has a pin delay value specified in the new pin delay format for any physical pin that is not mapped to it.

**SPLBPD-188, “No %s value specified in %s for physical pin %s. Ignoring this entry.”, W**

This warning is generated if a physical pin definition in the *Logical Pins* grid contains only the pin number and no property value.

**SPLBPD-199, “The cell cannot be saved because the value specified for the ALT\_SYMBOLS property is not syntactically correct. Use the browse button next to the Alt Symbols field on the General page of the Package Editor to specify the ALT\_SYMBOLS value and then save the cell.”, E**

This error message is generated when the value specified for the ALT\_SYMBOLS property is syntactically incorrect. The correct syntax is : (Subclass:Symbol ,... ; Subclass:Symbol ,...). For information on specifying values in the *Alt Symbols* field, see Browse Alt Symbol on page 436.

For more information on the ALT\_SYMBOLS property, see *Allegro Platform Properties Reference*.

**SPLBPD-200, "Specify a filename.",E**

This error is generated when a null value is specified for a filename.

**SPLBPD-201, "File %s does not exist. Check the filename and location.",E**

This error is generated when a specified file does not exist.

**SPLBPD-202, "File %s cannot be read. Check if the path and the extension are correct.",E**

This error is generated when a file cannot be found in the specified path or the extension is invalid for the operation. For example, for the *Open Project* operation, the specified file must have the `.cpm` extension.

**SPLBPD-203, "File %s is not accessible for writing. Change the permissions and try again.",E**

This error is generated when the specified file does not have write permission. Change the permissions and try again.

**SPLBPD-204, "Error on line %s of file %s. The file cannot be parsed. Correct the syntax error and try again.",E**

This error is generated in the following conditions:

- An error is found after the words MODEL, PACKAGE, PIN\_COUNT, or MODEL\_PORT while parsing a ptm file.
- The minimum columns required is set to the pin name or pin number column (whichever occurs last) and the data given under the header rows contains fewer number of columns than the required number of columns when parsing a CSV file.

For more information, see [Import Synopsys PTM Model](#) on page 255 and [Import Comma Separated Value \(.csv\) File](#) on page 246.

**SPLBPD-205, "The property value of a soft property can only be '?'. To accept any other value, set the Symbol\_ValidateSoftPropValue directive in the CPM file for your project to any value other than 1.",E**

This error is generated when a soft property has a value other than ?. You can configure Part Developer to accept any value as a soft property value by modifying the `Symbol_ValidateSoftPropValue` directive. For more information, see [Configuring to Accept ? as the Only Placeholder Property Value](#) on page 342.

**SPLBPD-206, "%s is a folder. Select a file.",E**

This error is generated if the filename specified while importing or selecting a model is not a valid filename.

**SPLBPD-207, “The compilation cannot be done because you have not specified the command for compilation. Specify the ncverilog command or the ncvhdl command.”**

This error message is generated if no command name is specified when verifying VHDL compilation or Verilog compilation using the *Tools – Verify* option. You must provide a command name such as `ncvhdl` or `ncverilog` to compile the VHDL or Verilog wrappers. For more information, see [VHDL Compilation](#) on page 316 and [Verilog Compilation](#) on page 316.

**SPLBPD-208, "PACK\_TYPE %s present on the symbol does not exist for logical part %s.",W**

This warning is generated when the value of the `PACK_TYPE` property for a symbol does not match any of the physical parts. For more information about the `PACK_TYPE` property, see [Allegro Platform Properties Reference](#).

**SPLBPD-209, "Part name %s defined on the symbol could not be found.",W**

This warning is generated when the value of the `PART_NAME` property on the symbol does not match the value of the `PART_NAME` field of the associated package, or any of the physical part names. For more information about the `PART_NAME` property, see [Allegro Platform Properties Reference](#).

**SPLBPD-210, "Package %s cannot be packaged with the symbol because the PACK\_TYPE property is defined incorrectly. Modify the physical part name displayed on the General page of the package.",W**

This warning is generated when the symbol is not packageable into the package indicated by the PACK\_TYPE property value.

For a list of supported characters in the PACK\_TYPE property, see [Supported Characters for PACK\\_TYPE Property Values](#) on page 563.

For more information about the PACK\_TYPE property, see *Allegro Platform Properties Reference*.

**SPLBPD-211, "The part cannot be verified using hlibftb because VALID\_PACK\_TYPE %s for the symbol does not exist for logical part %s. Modify the VALID\_PACK\_TYPE value either for the symbol or for the logical part to make sure that the symbol and the part are in sync. ",W**

This warning is generated when there is no physical part matching the value specified for the VALID\_PACK\_TYPE property.

**SPLBPD-212, "Wrong VALID\_PACK\_TYPE property defined. Package %s is not packageable with the symbol.",W**

This warning is generated when the symbol is not packageable into the package indicated by the VALID\_PACK\_TYPE property value.

**SPLBPD-213, "The PTF property without a value cannot be added because its context is set to global. Specify a value or change the context.",E**

This error is generated when a null value is specified for a PTF property with the context set to global. You can add PTF properties in the *PTF Properties* grid by choosing the *Tools – Setup – PTF* option.

**SPLBPD-214, "Pass-through pin(s) %s is (are) present in the Symbol Pins grid. Check if this is intended." W**

This warning is generated if a pin name is present more than once in the Symbol Pins grid. You can check if the specified pins are pass-through pins, which are optional symbol pins that make it easier to connect pins such as clocks, Chip Enable (CE), and select lines for a group of parts.

**SPLBPD-215, "The size of the symbol is greater than the maximum symbol size allowed. Reduce the symbol size by modifying the symbol outline or configure the Symbol\_MaxSymSize directive in the setup.cpm file according to your requirements.",E**

This error message is generated if the size of the symbol is greater than the maximum allowed size of the symbol. The maximum size is specified through the `Symbol_MaxSymSize` directive in the `setup.cpm` file. This file is stored at `<your_inst_dir>\share\cdssetup\LMAN`.

**SPLBPD-216, "The size of the symbol is greater than the sheet size. Choose Tools - Setup - Symbol to specify a larger sheet or reduce the size of the symbol by modifying the symbol outline.",E**

This error message is generated if the size of the generated symbol is greater than the sheet size specified through the *Tools – Setup – Symbol* option. For more information, see [Setting Up Symbol Properties](#) on page 93.

**SPLBPD-217, "Vector and sizeable pins are not allowed in the Global Pins section. To add a vector pin, choose Pins - Add from the Package Editor. To add a sizeable pin, choose Pins - Add from the Symbol Editor.", E**

This error is generated if you try to add a vector or sizeable pin in the Global Pins section. Only a logical pin or a symbol pin can be defined as vector or sizeable. For more information, see [Adding Logical Pins](#) on page 112.

**SPLBPD-221, "Could not load Si2 PinPak file %s. Incorrect format.",E**

This error is generated when the specified file does not follow the Si2 PinPak format standards or is an invalid XML file.

**SPLBPD-223, "Invalid pin type '%s' defined for pin %s. Converting it to UNSPEC.", W**

This warning is generated if the Ecix(Si) XML file being imported has a pin with pin type other than undefined, collector, emitter, ground, or unconnected and the direction value is any value other than I, O, or B. Such pin types are converted to UNSPEC.

**SPLBPD-226, "Empty property name found in the header. The corresponding values will not be imported. If this is not desired, add the property name in the header and reimport the file.", W**

This error message is generated in CSV import when a data column with no header name is found. Header names are mandatory for Part Developer to import part data from CSV files. For more information, see [Import Comma Separated Value \(.csv\) File](#) on page 246.

**SPLBPD-227, "Package pin properties attached to body section pin %s %s will be ignored during import.", W**

This warning is generated when pin properties are found on pins attached to the body section (POWER, GROUND, or NC). Such pin properties are ignored. Pin properties are valid if the pins are in the pin section of the chips.prt file. For more information, see *Allegro Design Entry HDL Libraries Reference*.

**SPLBPD-228, "The number of values entered on line(s) %s is less than the number of entries in the header row. Modify the import file to add the missing values.", W**

This warning is generated if the CSV file used in import has a row that has empty values for some properties specified in the header row. For more information, see [Import Comma Separated Value \(.csv\) File](#) on page 246.

**SPLBPD-229, "The number of values entered on line(s) %s is more than the number of entries in the header row. The extra values will be ignored. If this is not desired, modify the import file and add the missing header elements.", W**

This warning is generated when in an import data source, such as a CSV file, a header has fewer elements than the values specified under the header. For more information, see [Import Comma Separated Value \(.csv\) File](#) on page 246.

**SPLBPD-231, "CSV file %s cannot be imported because of incorrect file format. Make sure that the import file is a comma-separated text file.", E**

This error message is generated when the CSV file is formatted incorrectly. For more information, see [Import Comma Separated Value \(.csv\) File](#) on page 246.

**SPLBPD-234, "Could not load PTM file %s because of incorrect format. Correct the file and then reload.",E**

This error message is generated when the PTM file is formatted incorrectly. For more information, see [Import Synopsys PTM Model](#) on page 255.

**SPLBPD-236, "File %s is not valid for %s import because it does not have the required information. Make sure that the file has the required information in the correct format.",E**

This error message is generated when the import data source has invalid data.

**SPLBPD-237, "Property %s for pin %s will be ignored because of syntax errors. Specify the property in the correct syntax, %s.",E**

This error message is generated when a pin property is found to be syntactically incorrect. Such pin properties are ignored.

**SPLBPD-238, "Invalid property %s found for %s pin %s. Ignoring.",E**

This error message is generated if the CSV file being imported has a load property that is not in compliance with the pin type — such as a pin with type OC has the INPUT\_LOAD property value. You can see the valid load properties for each pin type through the *Tools – Setup – Package Pins* option.

**SPLBPD-239, "Line %s will not be imported because it contains a null value for the PIN\_NAME property. Correct the input file.",E**

This error is generated when a null value is found for the PIN\_NAME property during CSV import. For more information about the PIN\_NAME property, see *Allegro Platform Properties Reference*.

**SPLBPD-241, "Could not load EDA XML file %s because of incorrect format. Correct the file and then reload.",E**

This error message is generated when the EDA XML file is formatted incorrectly.

**SPLBPD-247, “Could not load ViewLogic file %s because of the unsupported block type at line %s”, E**

This error is generated when the ViewLogic file selected for import has a line that begins with Y but has a value other than 0,1, or 4. For more information, see [Import ViewLogic\(VL\) Part](#) on page 273.

**SPLBPD-253, “ECO process for cell %s of library %s from file %s is not successful”, E**

This error is generated whenever Part Developer encounters some problem like improper format of file and is unable to complete the ECO process.

**SPLBPD-254, "Duplicate pin name %s being converted to %s", W**

This warning message is generated when a duplicate pin name is found when importing data from the Si2 Pin Pak XML file. Such pin names are renamed to PINNAME\_1, PINNAME\_2, and so on.

**SPLBPD-257, "Duplicate property %s found on %s", W**

This warning message is generated when a duplicate package, symbol, package pin, or symbol pin property, such as the PIN\_TEXT property in the symbol.css file, the BODY\_NAME property in the chips.prt file, or the INPUT\_LOAD property for a pin in chips.prt, is found.

**SPLBPD-261, "Could not load Verilog file %s because of incorrect format. Correct the file and then reload.",E**

This error is generated when the Verilog file specified for import is formatted incorrectly. For more information, see [Import Verilog Model](#) on page 257.

**SPLBPD-264, "Could not load VHDL file %s because of incorrect format. Correct the file and then reload.",E**

This error message is generated when the VHDL file specified for import is formatted incorrectly. For more information, see [Import VHDL Model](#) on page 258.

**SPLBPD-266, “Model pin ‘%s’ is already mapped to another pin. Specify a unique pin number.”, E**

This error message is generated if two pins are mapped to the same model port in the vhdl.map or verilog.map file.

**SPLBPD-267, "XML export not successful",E**

This error message is generated when Part Developer fails to export part data to XML format. This may happen because a file with the same name already exists in the destination folder.

**SPLBPD-274, “Unable to check out license %s”, E**

This error message is generated when Part Developer fails to find any license files in the system. This may happen if the license server is not running or the CDS\_LIC\_FILE environment variable on your machine is not pointing to the correct license server. For more information, see *Cadence License Manager*.

**SPLBPD-275, "Unable to perform the operation because the %s feature is not supported in the license you are using currently. Choose File - Change Product to select the Allegro PCB Librarian XL license.",E**

This error is generated when a feature that is available only in the Allegro PCB Librarian XL licence is accessed using some other license. For more details, see [Features Available with Allegro Managed Library Authoring License](#) on page 19.

**SPLBPD-277, "Unable to load view %s because this feature is not supported in the license you are using currently. Choose File - Change Product to select the Allegro PCB Librarian XL license.",E**

Several views, such as simulation views, for a part are created only when Part Developer is launched using the PCB Librarian XL licence. This error is generated when an attempt is made to load such a part using a license other than the PCB Librarian XL license. For more details, see [Features Available with Allegro Managed Library Authoring License](#) on page 19.

**SPLBPD-279, “CSV export not successful”,E**

This error message is generated if the CSV file export fails for the selected package. This may happen if a file with the same name as the new file to be created through CSV export already exists.

**SPLBPD-284, “Export failed because the package is empty. Add pin information and then export the part.”,E**

This error message is generated if no pins are present in the part being exported in CSV format.

**SPLBPD-285, “Unable to change product as only one product present”, E**

This error is generated if only one license is available with the user and the *File – Change Product* option is used. For more information, see *Cadence License Manager*.

**SPLBPD-291, “The export process cannot be started because cell %s of library %s is not saved. First, save the cell and then restart the export process.”,E**

This error is generated when an attempt is made to export a part in Capture or Mentor format without first saving the modifications made on it.

**SPLBPD-294, “Pin(s) %s are mapped in wrapper %s for model %s but are missing on the symbol(s).”,W**

This warning is generated if some pins are present in the Verilog or VHDL wrapper and are mapped to model ports in the wrapper but these pin(s) do not exist on any of the symbols. For more information, see Chapter 9, “Working with VHDL Wrappers and Map Files” and Chapter 10, “Creating Verilog Wrappers and Map Files.”

**SPLBPD-295, “Pin(s) %s are not mapped in wrapper %s for model %s but are present on the symbol(s)”,W**

This warning is generated when a pin exists on the symbol and in the Verilog or VHDL wrapper, but is not mapped to any of the model ports in the wrapper. For more information, see Chapter 9, “Working with VHDL Wrappers and Map Files” and Chapter 10, “Creating Verilog Wrappers and Map Files.”

**SPLBPD-296, “BODY\_NAME property does not exist in package %. Since backannotation will not work without this property, it will be written automatically when the cell is saved.”,W**

The BODY\_NAME property is required for backannotation. The value of this property should match the cell name. This warning is generated when the property is not found in the package. On saving the part, the BODY\_NAME property is written automatically.

**SPLBPD-297, "The fixed-size symbol %s is not packageable with %s because the SIZE property value is greater than the number of slots. Reset the SIZE property to less than or equal to the number of slots.", E**

This error is generated when the `SIZE` property has a value greater than the number of slots for which a pin is present. For more information about fixed-size symbols, see [Creating Sizeable and HAS\\_FIXED\\_SIZE Symbols](#) on page 138.

**SPLBPD-298, "ViewLogic export cannot proceed because %s %s does not exist in the cell. Correct the input file and reimport.", E**

This error is generated when package or symbol information is not found for ViewLogic export.

**SPLBPD-300, "Package %s cannot be added in map view %s because the map file already has package information.", E**

This error is generated when a package name is copied and pasted in the map file that already has the package information.

**SPLBPD-301, "Logical pin %s mapped with model pin %s does not exist. Choose Pins - Global Delete to delete the logical pin.", E**

This error message is generated if a logical port is mapped to a model port in a Verilog wrapper but that logical pin is deleted from the package using *Delete Selected Rows* instead of the *Global Delete* option and then the cell is saved, closed, and re-opened.

**SPLBPD-302, "More than one %s section is present. Only the first one is read, rest ignored", W**

This warning is given if more than one instance are present in the `verilog.v` file for a Verilog wrapper. Only the first instance is read, and the rest are ignored. For more information, see [Chapter 10, "Creating Verilog Wrappers and Map Files."](#)

**SPLBPD-303, "Invalid Verilog wrapper. No instance section is present. Add only one instance.", E**

This error message is generated when in the wrapper file, no instance section is present. This may happen if the instance section is accidentally deleted from the `verilog.v` file or if any invalid wrapper file is used.

**SPLBPD-304, "Library %s cannot be accessed because it is not defined in the cds.lib file. From Project Manager, choose Tools - Setup, click Edit on the Global page, and define the library.",E**

This error is generated if any of the libraries specified in the .cpm file is not defined in the cds.lib file. For more information about the cds.lib file, see *Allegro Design Entry HDL Libraries Reference*.

**SPLBPD-305, "Annotated %s is empty. Specify a value for the generic or parameter annotated on the symbol.",E**

This error is generated if the Annotate option is checked in the Annotate Generics or Annotate Parameters dialog box, but the value to be annotated is left empty. For more information, see [Chapter 10, “Creating Verilog Wrappers and Map Files”](#) and [Chapter 9, “Working with VHDL Wrappers and Map Files.”](#)

**SPLBPD-307, “Duplicate base port name %s found. Ignoring the port.”, E**

This error is generated in the following conditions:

- Two pin names with the same base name exist in the CSV file that is being imported. For example, one pin is named AB<0> and the other is named AB and the pin type is POWER, GROUND, or NC for both.
- Port names are found to be same for the VHDL model being imported.

**SPLBPD-308, "Ignoring duplicate base pin name %s found in the cell. Correct the data to avoid losing one of the duplicate pins.",E**

This error message is generated when two pins are found with the same base name, such as AB<0> and AB.

**SPLBPD-309, "Functional group(s) %s of package %s is (are) not associated with any symbol due to logical pin mismatches in the chips and the symbol. Use the Interface Comparison tool to correct the problem.",E**

This error is generated in the following conditions:

- If the logical pin list of a functional group for a package does not match with any of the symbol pin lists. You can use the Interface Comparator to identify and fix the mismatch. For more information, see [Chapter 14, “Interface Comparator.”](#)

- If the value of the HAS\_FIXED\_SIZE property set on a symbol is different from the number of slots being represented by the symbol. The number of slots represented by the symbol is calculated by identifying the physical pin list of the common pin. For example, consider a symbol with the HAS\_FIXED\_SIZE property set to 4. Now, assume there is a common pin B with the following pin list (1,1,6,6,6). Now, since the common pin has a different physical pin number for slots 1 and 2, this symbol will have a mismatch with the value of the HAS\_FIXED\_SIZE property. The maximum value for the HAS\_FIXED\_SIZE property has to be less than or equal to the number of slots in which the common pin is present with unique physical pins. In the above example, the highest possible value for the HAS\_FIXED\_SIZE property is 3.

**SPLBPD-310, "No package is associated with symbol %s due to logical pin mismatches in the chips and the symbol. Use the Interface Comparison tool to correct the problem.",E**

This error message is generated in the following conditions:

- If the symbol cannot be packaged into any of the existing packages. You can use the Interface Comparator to identify and fix the mismatch. For more information, see [Chapter 14, "Interface Comparator,"](#).
- If the value of the HAS\_FIXED\_SIZE property set on a symbol is different from the number of slots being represented by the symbol. The number of slots represented by the symbol is calculated by identifying the physical pin list of the common pin. For example, consider a symbol with the HAS\_FIXED\_SIZE property set to 4. Now, assume there is a common pin B with the following pin list (1,1,6,6,6). Now, since the common pin has different physical pin numbers for slots 1 and 2, this symbol will have a mismatch with the value of the HAS\_FIXED\_SIZE property. The maximum value for the HAS\_FIXED\_SIZE property has to be less than or equal to the number of slots in which the common pin is present with unique physical pins. In the above example, the highest possible value for the HAS\_FIXED\_SIZE property is 3.

**SPLBPD-311, "Could not load view %s because it does not have the pdv.map file required to parse the VHDL wrapper view. Create the map file in Part Developer.", W**

This warning is generated if the `pdv.map` file is not found. For information on VHDL wrappers and map files, see the *Creating VHDL Wrappers and Map Files* chapter.

**SPLBPD-319, “Pin type %s changed to NC for pin NC because Part Developer requires logical pins with pin name NC to have the NC pin type. To use the specified pin type, change the pin name.”, W**

This warning is generated if a pin name NC has some pin type other than NC in the ViewLogic file that is being imported. The pin type is automatically changed to NC by Part Developer. This is done because Cadence flows require that any pin named NC must be of type NC.

**SPLBPD-320, “Converting scalar %s pin %s to a vector pin in the logical section because the pin is mapped to different physical pin numbers. If this conversion is not desired, correct the input file.”, W**

This warning is generated in the CSV file being imported if a scalar pin having type POWER, NC, or GROUND is present and another pin exists with same name, pin type and on the same symbol. Part Developer converts such pins to vector pins.

**SPLBPD-321, "Ignoring duplicate pin %s during import. If this is not desired, correct the input data.",W**

This warning is generated if in the CSV file being imported pins with type POWER, NC, or GROUND are present and if a duplicate pin name exists with same name and:

- The pin is a vector pin with the same type and on the same symbol.
- The two pins have different types.
- If no information is available about the symbol and the pins are vector pins with the same name.

**SPLBPD-322, "Body section pin %s of type NC renamed to NC. If this is not desired, change the pin type in the input file and reimport.",W**

This warning is generated when during CSV import, NC pins are found with names other than NC. Such pins are renamed to NC. Cadence flows require that NC pins in the body section must necessarily be named NC.

**SPLBPD-323, "The cell cannot be saved because pin(s) %s is (are) placed on the symbol origin. Move the symbol pin(s) away from the origin and then save the cell. ",E**

This error is generated if a pin is placed on the symbol origin, which implies that it has 0,0 as the value of the X,Y coordinates. Having 0,0 as the X,Y coordinates disables the access to the symbol-level properties in Allegro Design Entry HDL.

**SPLBPD-326, "Cell %s already exists. As a result, import of %s failed. Specify a different name for the cell or change the name of the input file. If you are using the command line, use the -overwrite option.",E**

This error is generated if an existing cell name is specified when creating a cell from import data.

**SPLBPD-327, "Ignoring the duplicate setup %s property with name %s and value %s because only one value for a given property is stored. Make sure that the correct value is saved.",E**

This error is generated when same setup directives exist in both 15.x and pre-15.x syntax. This might happen if you have a 15.x CPM file and a pre-15.x CPM file in the Cadence Search Functions (CSF) path and both files have the same property.

**SPLBPD-342, "Could not load ViewLogic file %s due to incorrect format. Make sure that the input file is a valid ViewLogic file.",E**

This error is generated if the file specified for Import ViewLogic or ECO ViewLogic is not a valid ViewLogic file.

**SPLBPD-344, "ViewLogic export not successful",E**

This error is generated if Part Developer fails to export the ViewLogic part.

**SPLBPD-350, "Ignoring invalid %s property %s found in the template. If this is not desired, correct the template.",E**

This warning is generated when an invalid property is found in the template. The predefined list of reserved properties are available in the `propfile.prop` file located at `<cds_inst_dir>\share\cdssetup\LMAN`. The reserved properties are listed using the `Not Allowed` keyword. For example, the reserved properties for a package are listed as:

```
(NotAllowed  
"ALT_SYMBOLS,CLASS,JEDEC_TYPE,NC_PINS,PART_NAME,PHYS_DES_PREFIX,POWER_PINS,SWAP_INFO")
```

It is strongly recommended that you do not change the predefined list of reserved properties as they have special handling in Cadence flows and changing them may have unpredictable behavior. To know more about the reserved properties and their use, see *Allegro Platform Properties Reference*.

**SPLBPD-351,"File %s cannot be parsed due to syntax errors. Either correct the syntax error in the file or use Part Developer to recreate the file.",E**

This error is generated in the following conditions:

- Some exception occurs when reading the symbol.css file
- Syntax error (starting '(' ) is not found while parsing the pinlist.txt file.

**SPLBPD-352,"Function group %s cannot be exported because no symbol is associated. Click Generate Symbol(s) to create a symbol and then start ViewLogic export.",E**

This error is generated for those functional groups in a package that are not associated with any symbols in the cell and this package is selected for ViewLogic export. The unassociated functional groups will not be exported.

**SPLBPD-356,"Could not load the Mentor part from component directory %s and map file %s due to incorrect format. Make sure that the input file is a valid Mentor file.",E**

This error is generated when the *Import Mentor* part option is selected with the *Choose from Component Directory* option and some error occurs while parsing the map file.

**SPLBPD-358, "Mentor export not successful. Check if the specified directory already contains a part with the same name as the cell.",E**

This error is generated when Mentor export fails due to some reason, such as a cell with the same name as the one being exported already exists in the directory specified for export.

**SPLBPD-366, "Could not import FPGA files because they are not of supported format. Make sure that the input files are valid and supported FPGA files.",E**

This error is generated when Part Developer encounters an error, such as an invalid FPGA file, when importing FPGA files with the *Use standard component* option.

**SPLBPD-368, "The voltage value is not same for all occurrences of pins %s. If this is not desired, make sure that the input file is correct.",W**

This warning is generated when the voltage value is not same for all occurrences of some pins in the FPGA files (Xilinx and Altera Quartus) being imported.

**SPLBPD-369, "No pin information could be extracted from FPGA file %s. Make sure it is a valid %s file.",E**

This error is generated in the Import FPGA process when a file cannot be parsed for pin information. The parsers used are based on the vendor name that the user selects from the drop-down list in the Import FPGA panel.

**SPLBPD-370, "Property %s will be added with default values to pin %s of type %s because the property is missing in chips.prt. Check the validity of pin types before saving the cell.",W**

This warning is generated if one of the required properties for a pin type is missing from the chips.prt file. Part Developer will automatically add the missing property with the default values. For example, if for a pin type OC\_BIDIR, the OUTPUT\_LOAD property is missing, Part Developer will automatically add this property to the pin and give the warning.

**SPLBPD-371, "Invalid property %s deleted from pin %s of type %s. Check the validity of pin types before saving the cell.",W**

This warning is displayed when Part Developer removes an invalid property from a pin. For example, if a pin in the chips.prt file is of type such as ANALOG but has the OUTPUT\_LOAD\_VALUE or INPUT\_LOAD\_VALUE property, this property is removed and a warning is displayed.

**SPLBPD-372, "The cell cannot be baselined because there are errors in the cell. Resolve the problems and then retry baselining the cell.",W**

This warning is displayed when a cell has errors and a save is attempted with the *Baseline on Save* option enabled. See the Errors and Warnings dialog box for details of the errors in the cell.

**SPLBPD-375, "Could not read Mentor catalog file %s due to incorrect or unsupported format. Either correct the catalog file(s) or select the map file and the symbol to import the part.",E**

This warning is given if Part Developer is not able to complete the reading of Mentor catalog when importing Mentor parts.

**SPLBPD-376, "Mentor DDP Error during import/export: %s",E**

This warning is generated if some DDP error occurs during importing from Mentor files or exporting to Mentor files.

**SPLBPD-377, "Could not load Mentor library file %s. Make sure that this file is present at \$MGC\_HOME/lib and the version is correct.",E**

This error is generated if some file(s) from the Mentor library at `MGC_HOME\lib` cannot be accessed.

**SPLBPD-382, "Could not load APD files from directory %s due to incorrect format. Make sure that the complete set of component files created by APD is present in the same directory.",E**

This error is generated in the following conditions:

- Some files in the component directory are not in a format required for APD import
- Only the *ECO only pin delay values* option is selected and some files in the component directory are not in a format required for APD import.

**SPLBPD-386, "Could not import the selected footprint.",E**

This error is generated if a specified footprint file could not be imported.

**SPLBPD-388, "Footprint %s not present in PSMPATH. Specify only the footprints present in PSMPATH or update PSMPATH.",E**

For information on how to update PSMPATH, see [Modifying Footprint Information](#) on page 161.

**SPLBPD-403, "Project %s is read-only. Reload the project after getting write permissions if you want to create new cells or modify existing cells.",E**

This error is generated when the selected CPM file is read-only. Change the permissions and try again.

**SPLBPD-404, "Unable to create the temp directory because of no write permissions in %s. Change the permissions and then retry the operation.",E**

This error is generated when Part Developer is unable to create the temp directory because the directory in which the project file is present is read-only.

**SPLBPD-411, "The cell cannot be saved because symbol sym\_1 is not present in the cell. If a cell has a symbol view, make sure that the symbol names are in a sequence starting from sym\_1.",E**

This error is generated when any symbol exists for the cell but sym\_1 is not present on the cell and the cell is saved. Cadence flows require that sym\_1 be present in cells that have the symbol view.

**SPLBPD-413, "Ignoring line %s of file %s because of incorrect syntax. Check the symbol pins and graphics for missing information.",E**

This warning is given for the lines in the `symbol.css` file that were ignored because of missing line identifiers. For more information on `symbol.css`, see *Allegro Design Entry HDL Libraries Reference*.

**SPLBPD-414 "Ignoring line %s of file %s because of extra or missing parameters. Check if the file has a valid format.",E**

This error is generated if some mandatory parameters are missing in a map file or some extra parameters are present in addition to mandatory parameters.

**SPLBPD-415, "Line %s of file %s is not complete. To correct the problem, save the symbol.",W**

This warning is generated when Part Developer corrects the `symbol.css` file because the lines starting with L, M, A, T, X, C, or P are not complete or some optional parameters are missing.

**SPLBPD-416, "The part cannot be used in the flow because vector pins with basename %s have different bit orders. Modify the bit information to ensure that either the 0..n format or the n..0 format is followed.",E**

This error is generated if vector pins have different bit orders, such as 0..2 and 2..0, in two symbols and the cell is saved or opened.

**SPLBPD-417, "HAS\_FIXED\_SIZE property value is incorrect for symbol %s. The correct value will be added during save.",W**

This warning is generated if the HAS\_FIXED\_SIZE property value in symbol.css is different from the size of the sizeable pin that has been set using the *Set Size* option.

**SPLBPD-418, "HAS\_FIXED\_SIZE property was missing or was incorrect for symbol %s. Correct value has been added.",W**

This warning is generated if a sizeable pin is present in a symbol and you try to add the size to the symbol for the first time or modify the size using the Set Size option.

**SPLBPD-450, "Could not load Capture package %s. The Capture library might be corrupt. Check the part in Capture and then retry importing.",E**

This error message is generated if the Capture library is corrupt.

**SPLBPD-452, "Ignoring line %s because of syntax errors in the property definition. If this is not desired, correct the input file and reimport.",E**

This error is generated in the following conditions:

- During the import of an APD file, a property name and value are present and additional two or more values are present in the same row in the RPT file.
- During the import of a CSV file, a property name is missing or a property value is missing or there is extra data in the properties segment (a third non-empty column exists).

**SPLBPD-453, "The setup file '%s' could not be loaded. Check if the file exists. Alternatively, you can modify the value of the load\_setupfile property in the input file to specify another filename.",E**

This error is generated in the following conditions:

- The load\_setupfile property exists in the APD file being imported and the setup file specified as the value to this property cannot be opened.
- The load\_setupfile property exists in the CSV file being imported and the setup file specified as the value to this property cannot be opened.

## Part Developer User Guide

### Errors and Warnings

---

#### **SPLBPD-454, "Invalid pin property %s will be ignored. If this is not desired, correct the input file and reimport.",E**

This error is generated if a pin property name in the header row contains invalid characters in the CSV file to be imported. The list of invalid characters is as follows:

- ~
- `
- !
- \*
- (
- )
- -
- +
- =
- |
- \
- }
- ]
- {
- [
- :
- ;
- "
- ,
- <
- >
- .
- ?

**SPLBPD-455, "Import cannot be performed because the required header pin\_name is missing. Correct the input file and reimport.",E**

This error is generated if the `pin_name` entry is missing in the header row of the CSV file to be imported. For more information, see [Import Comma Separated Value \(.csv\) File](#) on page 246.

**SPLBPD-456, "Import cannot be performed because the required header pin\_number is missing. Correct the input file and reimport.",E**

This error is generated if the `pin_number` entry is missing in the header row of the CSV file to be imported. For more information, see [Import Comma Separated Value \(.csv\) File](#) on page 246.

**SPLBPD-457, "Line %s cannot be imported because the symbol number specified on the line is not a natural number. Make sure that the symbol numbers in the input file are assigned values greater than 0 and then reimport.",E**

This error is generated if a symbol property exists in the header for the CSV file being imported and its value is not a natural number.

**SPLBPD-458, "Ignoring property '%s' because it has an invalid value. If this is not desired, correct the input file and reimport.",E**

This error is generated if any of the following properties exists in a CSV file being imported and the property values are invalid:

- NO\_LOAD\_CHECK
- NO\_IO\_CHECK
- ALLOW\_CONNECT
- UNKNOWN\_LOADING
- NO\_ASSERT\_CHECK
- NO\_DIR\_CHECK

**SPLBPD-459, "Import cannot be performed because no appropriate header row is found in the input file. Modify the input file to add supported headers and then reimport.",E**

This error is generated when no header row is found in the CSV file being imported. For more information, see [Import Comma Separated Value \(.csv\) File](#) on page 246.

**SPLBPD-460, "Import cannot be performed because symbol number %s is missing. Make sure that the symbol numbers in the input file are in a sequence.",E**

This error is generated when the symbol property exists in the header and a symbol number is missing from the values in the CSV file that is being imported. For example, if symbol numbers are 1, 3, and 4 and 2 is missing from the values, this error message is generated.

**SPLBPD-461, "Import failed because of an error on line %s and token '%s' in file %. Correct the input file and reimport.",E**

This error is generated when some syntactical error (like a missing or extra ; ) is found in some VHDL file or Verilog file.

**SPLBPD-463, "Capture read failed for %. Check if the Capture library is corrupt.",E**

This error is generated if the Capture library is corrupt.

**SPLBPD-469, "The tool setup could not be loaded because of syntax errors. Correct the setup information.",E**

This error is generated if the `setup.cpm` file could not be loaded from the `<cds_inst_dir>\share\cdssetup\lman` location when starting Part Developer.

**SPLBPD-470, "Ignoring invalid value '%s' for the CLASS property. Redefine the CLASS property value by choosing a valid value from the Class drop-down list on the General page.",E**

This error is generated if the CLASS property has some invalid value in the CSV file that is being imported. Before importing, you can modify the predefined list of valid CLASS property values through the `PackageClass` keyword in the `profile.prop` file located at `<cds_inst_dir>\share\cdssetup\LMAN`. Alternatively, you can import and then choose a valid value from the *Class* drop-down list on the General page.

**SPLBPD-471, "The model and the symbol cannot be bound because the MODEL\_BINDING property value is not specified in the correct format. Specify the value in libname.cellname:viewname format in the map file and reload the part.",E**

This error is generated if *Bind Model and symbol* is used for a VHDL model or wrapper and the value of the MODEL\_BINDING property existing in the vhdl.map file is specified in an incorrect format, such as . and : are not properly placed. The correct format for specifying the MODEL\_BINDING property value is libname.cellname:viewname.

**SPLBPD-472, "%s",E**

This error is generated if some DDB error occurs when reading a PTF file. The message displayed is the error or warning reported by DDB.

**SPLBPD-473, "%s",E**

This error message is the exception message displayed if a file could not be opened or created.

**SPLBPD-475, "DML parser: %s",W**

This warning is generated if the DML parser has found some errors.

**SPLBPD-476, "DML parser: %s",E**

This error is generated if the DML parser has found some errors. The reported error is given in the message itself.

**SPLBPD-479, "Could not read model/primitive %s from file %s",E**

This error is generated if Part Developer fails to read the model or primitive information from the specified file.

**SPLBPD-482, "Could not read model/primitive list from file %s",E**

This error is generated if Part Developer fails to read the model or primitive list from the specified file.

**SPLBPD-483, “Import cannot proceed because the package family is not same as the selected family. Correct the input file and reimport.”, E**

This error is generated in FPGA import if the selected family is not same as the family specified in the Actel file.

**SPLBPD-484, “SIZE property value is incorrect for symbol %s. The correct value will be added during save.”, W**

For more information, see [Creating Sizeable and HAS\\_FIXED\\_SIZE Symbols](#) on page 138.

**SPLBPD-505, “For logical part %s, pack type %s is present in the chips.prt file but missing in the part table file. Add PTF rows for that package to ensure that you can instantiate the package.”, W**

This warning is generated when the pack type is present in the package but missing from the corresponding part table file.

**SPLBPD-506, “For logical part %s, pack type %s is missing in chips.prt but present in the part table file. The default package will be assigned in chips.prt during packaging. If this is not desired, specify the pack type on the General page of the Package Editor.”, W**

This warning is generated when the pack type is present in the part table file but missing from the corresponding package.

**SPLBPD-522, “Import cannot proceed because the specified model does not have any valid pinmap section. Make sure that the input file is valid and complete.”, E**

This error is generated when the selected DML or IBIS file does not have the component name.

**SPLBPD-524, “Could not load DML file %s due to incorrect import file format. Make sure that the input file is a valid DML file.”, E**

This error is generated when the selected DML file is not formatted correctly or is not a valid DML file.

**SPLBPD-527, "The SI model interface comparison cannot be done because you have not specified the object(s) to be compared. After selecting the types of both objects, specify the objects in the empty fields and then click Compare.",E**

This error is generated when the primitive field is empty in the selected IBIS, DML, chips, or footprint file.

**SPLBPD-529, "Some of the programs required for this functionality are not installed. Install product PX3120 Allegro PCB Model Integrity for the missing programs.",E**

This error is generated when `IBIS2CON` and `DML2CON` translators cannot be run because model integrity tools are not available. Install product PX3120 Allegro PCB Model Integrity for the missing programs.

**SPLBPD-532, "Invalid syntax for physical pin property %s on line %s. The line will be ignored. If this is not desired, correct the property definition.",W**

This warning is generated when a syntax error is encountered in the definition of a physical pin property, such as `PIN_DELAY`, in the `chips.prt` file. The syntax error could be due to a missing `PIN_DELAY` value or the use of token `:` more than once before `'.'`. Part Developer throws this warning and ignores the line having the error.

**SPLBPD-533, "Ignoring property %s from the body section because physical pin properties are not supported in this section.",W**

This warning is generated if physical properties that are not supported in the body section of the `chips.prt` file are specified in that section. Part Developer throws this warning and ignores the lines. For example, in import CSV, if the `PIN_DELAY` value is attached to global pins, the value will be ignored.

**SPLBPD-535, "Ignoring the duplicate physical pin property %s found for physical pin %s. If this is not desired, reenter the value.",W**

This warning appears when a duplicate physical pin property is found for physical pins.

**SPLBPD-701, "Import cannot proceed due to the invalid range specified for import. Make sure that the number of rows to import is a positive integer and less than %s rows available for import.",E**

This error is generated during text import if the start row number or the end row number is negative or a non-integer value and if the number of lines to import exceeds the number of lines in the text file.

**SPLBPD-703, "Could not import the text file because no pin\_name column is present. Either modify the text file to add a pin\_name column or select the Set pin number as pin name check box on the Select Columns page.",E**

This error is generated if the text file to import does not contain the `pin_name` column, which is a required column for text import.

**SPLBPD-705, "Import cannot proceed because column name %s exists more than once in the data being imported. Modify the import data to ensure only one occurrence of each column exists.",E**

This error is generated when the data filtered for text import contains more than one column with the same name.

**SPLBPD-706, "Import cannot proceed because no column has been selected for import. Use the drop-down list in the grid header to select a column.",E**

This error is generated if the *Ignore* option is selected for all the columns displayed in the Data Preview area of the Select Columns page when importing a text file. For successful text import, at least the `PIN_NAME` column must be selected.

**SPLBPD-707, "Import failed because an invalid value is specified for directive %s in the profile file. Specify the valid value, %s.",E**

This error is generated if any directive is specified a wrong value in the profile file used for text import. For more information on profile file directives, see [Profile Use Model](#) on page 269.

**SPLBPD-709, "Could not write profile file %s. Check if the path is correct and you have write permission.",E**

This error is generated if the profile file specified for saving text import settings could not be written successfully due to an invalid path or nonavailability of write permission.

**SPLBPD-711, "Import cannot proceed because no column has been selected for import. Verify that you are using the correct profile file.",E**

This error is generated if no PIN\_NAME column can be extracted from the profile file used for text import. To prevent this error, you can choose *No* when asked to go to preview directly after loading the profile and verify the profile settings first and then use the profile to derive data from the text file. If the profile settings are not according to your requirements, you can use another profile or specify and save new profile settings. When creating a profile, remember that any selection made in the *Data Preview* area by using the pop-up menu is not saved in the profile file.

**SPLBPD-715, "The Pin Number column cannot be imported because you have selected the Set pin number as pin name check box. To proceed, either uncheck the Set pin number as pin name check box or choose Ignore from the drop-down list in the Pin Number column.",E**

Selecting the *Set pin number as pin name* check box ensures successful import even if the file being imported does not have a pin\_name column, which is a required column for text import. If the text file you are importing contains both pin\_name and pin\_number columns and you want to import both the columns, make sure that the *Set pin number as pin name* check box is not selected.

**SPLBPD-716, "Physical pins %s are present in footprint %s but not in the primitive of the PTF row with key %s. Add the missing pins by editing the primitive in the Package Editor or replace the footprint with a footprint that matches the primitive.",E**

For information on adding physical pins, see [Modifying Pin Lists and Mapping](#) on page 163.

For information on specifying another footprint, see [Modifying Footprint Information](#) on page 161.

**SPLBPD-717, "Function Group %s of package %s of the ptf row with key %s is not associated with any symbol due to logical pin mismatch in the chips and the symbol. To synchronize the package and the symbol, run the Interface Comparator.",E**

For information on how to synchronize the package and the symbol, see the *Interface Comparator* chapter.

**SPLBPD-800, "Both the connection point and anchor point for shape %s lie at the same X coordinate value. Reset the values properly and then save.",E**

This error is generated if the X coordinates of the connection point and anchor point for a pin shape are the same and an attempt is made to save the shape without setting the connection point or the anchor point such that these points lie in a straight line perpendicular to the axis.

**SPLBPD-802, "Both the connection point and anchor point for shape %s lie at the same Y coordinate value. Redefine the points to align them properly.",E**

This error is generated if the Y coordinates of the connection point and anchor point for a pin shape are the same and an attempt is made to save the shape without setting the connection point or the anchor point such that these points lie in a straight line perpendicular to the axis.

**SPLBPD-805, "Creation of a new shape, %s, failed. Check that the save path is not empty and you have the required permissions.",E**

This error is generated when a shape cannot be created because the save path is empty or incorrect or write access is not available.

**SPLBPD-806, "Error reading shape file %s. Make sure that the file has the correct format or does not contain invalid characters.",E**

This error is generated if Part Developer encounters an error when reading a shape file and therefore fails to display a specified shape. This error is likely to occur in situations where a shape file has been modified manually. To fix the problem, you should re-extract the shape using the Symbol Editor.

**SPLBPD-807, "The default shape save path or the shape path can be changed only when the Shape Editor is closed. Close the Shape Editor and try again.",E**

This error is generated if an attempt is made to modify the path fields in Setup without closing the Shape Editor.

**SPLBPD-808, "Extraction of a new shape, %s, failed. Check that the save path is not empty and you have the required permissions.",E**

This error is generated when a shape cannot be extracted from a symbol because the save path is empty or incorrect or write access is not available.

**SPLBPD-814, "Opening of shape %s failed. Make sure the you have the required permissions.",E**

This error is generated when a shape cannot be opened because read access is not available.

**SPLBPD-820, "Could not get the shape list due to some internal error. Check that the files in the read path are not hidden.",E**

This error is generated when the shape list cannot be retrieved and, as a result, the Shape Viewer cannot be populated.

**SPLBPD-823, "Deleting of shape %s failed. Make sure that you have the required permissions.",E**

This error is generated when a shape cannot be deleted because write access to the shape file is not available.

**SPLBPD-829, "Renaming of shape %s failed. Make sure that you have the required permissions.",E**

This error is generated when a shape cannot be renamed because write access to the shape file is not available.

**SPLBPD-831, "The shape file could not be accessed. Check the file permissions and try again.",E**

This error is generated when a shape file cannot be displayed because the shape file is not accessible.

**SPLBPD-832, "A shape with the name %s already exists or the name specified is the name of a standard shape. Specify a different name for the new shape.",E**

This error is generated if an attempt is made to save a new shape with a name that is same as the name of an existing shape or a standard shape. Part Developer does not support duplicate shape names.

**SPLBPD-833, "No shape was selected for extraction. Make sure you select all of the components of the shape.",E**

This error is generated when shape extraction fails because a shape is not selected. In order to ensure that a shape is extracted properly, you should make sure that you have selected all of the components of the shape.

**SPLBPD-834, "Shape %s could not be saved. Check the shape file permissions and try again.",E**

This error is generated when a shape cannot be saved in the specified file because write access to the file is not available.

**SPLBPD-835, "Shape %s could not be deleted. Verify that the shape exists and you have the required permissions.",E**

This error is generated when a shape cannot be deleted because a shape with the specified name does not exist or the required permissions are not available.

**SPLBPD-836, "The Shape field cannot be empty. Specify a valid name for the shape and then click OK.",E**

This error is generated when an attempt is made to save a shape without specifying a name. A shape name must be unique and should not contain any invalid character. The list of valid characters is as follows:

- A-Z
- 0-9
- -

**SPLBPD-837, "The Shape Save Path field is not specified or contains invalid values. Verify that the default shape save path is set in Tools – Setup – Shape and the path is valid.",E**

This error is generated while creating a shape if the specified save path does not exist. For more information on how to set the default shape save path, see [Setting Up a Project for Shapes](#) on page 181.

**SPLBPD-838, "No shape is selected. Select a shape and then try again.",E**

This error is generated when a shape is not selected.

**SPLBPD-839, "The Shape Path and Shape Save Path fields contain invalid values. Specify valid paths in each field and reload the project.",E**

This error is generated if the read path specified for a shape does not exist. For information on how to set the default shape save path, see [Setting Up a Project for Shapes](#) on page 181.

**SPLBPD-841, "Invalid character(s) %s found in shape name. Rename the shape using valid characters only.",E**

This error is generated while creating, renaming, and extracting a shape if the shape name contains invalid characters. The list of valid characters is as follows:

- A-Z
- 0-9
- -

**SPLBPD-842, "The pin shape attached with pin %s cannot be attached to the %s of the symbol. Check the location attribute for the shape and try again.",E**

This error is generated when an attempt is made to attach a pin shape to a side of the symbol other than the sides specified on the Shape Attributes pane.

**SPLBPD-843, "Shape(s) %s cannot be found. Therefore, the standard Line shape is attached to symbol pin(s) %s. If this is not desired, make sure that the missing pin shapes are available in the specified shape path.",W**

This warning is generated when the Import and Export wizard uses the standard *Line* pin shape instead of a pin shape specified in the file being imported because the specified shape cannot be found at the location pointed by the shape path. To recreate the symbol with the required pin shape, make sure that the shape exists and the shape path specified using *Tools – Setup – Shape* points to the right location.

**SPLBPD-850, "The Design Entry HDL schematic grid and the Part Developer symbol grid are different. Make sure that the symbol grid equals to the schematic grid.",E**

If your Part Developer symbol grid setting does not match the Design Entry HDL schematic grid setting, symbol pins can be off grid when the symbols are placed in the schematic. For information on specifying the Part Developer symbol grid setting, see [Pin grid size](#) on page 93.

**SPLBPD-851, "Part %s will not work in the flow because it contains a null key property, %s. Assign a property value through the Property Editor.",E**

This error message appears when the Instantiation and Packaging check is performed on a part that contains a null key property. To correct the problem, specify a value to the property either in the `chips.prt` file or through the Property Editor.

**SPLBPD-852, "Part %s will not work in the flow because of the invalid CLASS value %s in package %s. Modify the package information to ensure that the CLASS value is either ? or a value from the PreDefined list defined in the PackageClass section of propfile.prop.",E**

This error message appears when the Instantiation and Packaging check is performed on a part that contains an invalid CLASS value in a package. The CLASS property value can be either of the following:

- ?
- A value from the PreDefined list in the PackageClass section of `propfile.prop` located at `<cds_inst_dir>\share\cdssetup\LMAN`

**SPLBPD-853, "Part %s will not work in the flow because package %s and the PTF file have different CLASS property values. Modify either the package information or the part table information to ensure that there is no CLASS property mismatch between the package and the PTF file.",E**

This error message appears when the Instantiation and Packaging check is performed on a part that has different CLASS property values in a package and the PTF file. To correct the problem, do either of the following:

- Modify the package information through the Package Editor or by manually editing the `chips.prt` file
- Modify the part table information through the Package Editor or by manually editing the PTF file

**SPLBPD-854, "Part %s will not work in the flow because part type name %s defined in PTF has a length greater than that set through the PackagerXL CPM directive PART\_TYPE\_LENGTH. Modify the part type name so that the name is within the specified limit or redefine the PART\_TYPE\_LENGTH value in cds.cpm or in the project CPM file.",E**

This error message appears when the Instantiation and Packaging check is performed on a part for which the length of the part type name exceeds the limit set in the `cds.cpm` file or the `project.cpm` file through the PackagerXL CPM directive `PART_TYPE_LENGTH`.

To correct the problem, do either of the following:

- Modify the PTF file to specify a name within the limit
- Increase the number of characters specified in the `PART_TYPE_LENGTH` directive

**SPLBPD-855, "Part %s will not work in the flow because the length of the LOCATION property value in symbol %s is greater than the value set through the PackagerXL CPM directive REF\_DES\_LENGTH. Modify the LOCATION property value so that the length is within the specified limit or redefine the REF\_DES\_LENGTH value in cds.cpm or in the project CPM file.",E**

This error message appears when the Instantiation and Packaging check is performed on a part that has a symbol in which the length of the `LOCATION` property value exceeds the limit set in the `cds.cpm` file or the `project.cpm` file through the PackagerXL CPM directive `REF_DES_LENGTH`.

**SPLBPD-856, "Part %s will not work in the flow because a CLASS property value is not found in package %s. Modify package information to specify a valid value for the CLASS property.",W**

This error message appears when the Instantiation and Packaging check is performed on a part that has a null `CLASS` property value.

The `CLASS` property value can be either of the following:

- ?
- A value from the PreDefined list in the `PackageClass` section of `profile.prop` located at `<cds_inst_dir>\share\cdssetup\LMAN`

## Part Developer User Guide

### Errors and Warnings

---

**SPLBPD-857, "Part %s will not work in the flow because a CLASS property value is not found in package %s and PTF. Make sure that the same value is specified in both places.",W**

This error message appears when the Instantiation and Packaging check is performed on a part that does not have a CLASS property value specified in both a package and the PTF file. Modify package and part table information to add a same CLASS property value in both places.

**SPLBPD-858, "Part %s will not work in the flow because symbol %s is not associated with any package. In the symbol view, right-click on the symbol name and choose Generate Package to create an associated package.",E**

This error message appears when the Instantiation and Packaging check is performed on a part that has a symbol that is not associated with any package. All symbols except documentation symbols must be associated with packages.

**SPLBPD-860, "Part %s will not work in the flow because the PTF file contains an invalid CLASS value, %s. Make sure that the CLASS value is either ? or a value from the PreDefined list in the PackageClass section of propfile.prop.",E**

This error message appears when the Instantiation and Packaging check is performed on a part that contains an invalid CLASS value in the PTF file. The CLASS property value can be either of the following:

- ?
- A value from the PreDefined list in the PackageClass section of propfile.prop

**SPLBPD-900, "The pin order for the current package could not be saved. Save the cell and set the pin order again.",E**

This error message appears when the *Set Pin Order* option is chosen but the pin order cannot be saved.

**SPLBPD-950, "Differential pair %s cannot be created because positive pin %s is of pin type %s. Reassign the pin type(s) to ensure that both pins have the same pin type and the pin type is not GROUND, POWER, or NC.",E**

This error message appears when loading a part if the part has a GROUND, POWER, or NC pin with the DIFF\_PAIR\_PINS\_POS property. Part Developer creates a differential pair only if both the constituent pins have any pin type other than GROUND, POWER, and NC.

## Part Developer User Guide

### Errors and Warnings

---

**SPLBPD-951, "Differential pair %s cannot be created because negative pin %s is of pin type %s. Reassign the pin type(s) to ensure that both pins have the same pin type and the pin type is not GROUND, POWER, or NC.",E**

This error message appears when loading a part if the part has a GROUND, POWER, or NC pin with the `DIFF_PAIR_PINS_NEG` property. Part Developer creates a differential pair only if both the constituent pins have any pin type other than GROUND, POWER, and NC.

**SPLBPD-952, "Differential pair %s has positive pin %s and negative pin %s in different sections. If this is not intended, move one of the pins to ensure that both pins are in the same section.",W**

This warning message appears if you try to save a split part that has the constituent pins of a differential pair distributed across sections.

**SPLBPD-954, "Differential pair %s cannot be created because positive pin %s and negative pin %s have different pin types. Reassign the pin type(s) to ensure that both pins have the same pin type and the pin type is not GROUND, POWER, or NC.",E**

This error message appears when loading a part if the constituent pins of a differential pair do not have the same pin type. After the part is loaded, you can change the pin type, select the two pins, and choose *Create Differential Pair* from the shortcut menu.

**SPLBPD-955, "Differential pair %s cannot be created because pin %s is also defined as the positive pin of this pair. Make sure that only two pins are assigned the DIFF\_PAIR\_PINS\_POS and DIFF\_PAIR\_PINS\_NEG properties with the same differential pair name.",E**

This error message appears when loading a part if multiple pins are assigned the `DIFF_PAIR_PINS_POS` property with the same differential pair name. A differential pair can have only one positive pin and one negative pin.

**SPLBPD-956, "Differential pair %s cannot be created because multiple pins, %s, are defined as negative pins of this pair. Make sure that only two pins are assigned the DIFF\_PAIR\_PINS\_POS and DIFF\_PAIR\_PINS\_NEG properties with the same differential pair name.",E**

This error message appears when loading a part if multiple pins are assigned the `DIFF_PAIR_PINS_NEG` property with the same differential pair name. A differential pair can have only one positive pin and one negative pin.

**SPLBPD-957, "Differential pair %s cannot be created because the positive pin definition is missing or incorrect. Select the two pins and choose Create Differential Pair from the shortcut (RMB) menu. If you do not want to create the differential pair, remove the DIFF\_PAIR\_PINS\_NEG property from the negative pin in chips.prt.",E**

This error message appears when loading a part if a negative pin definition is found and the corresponding positive pin definition is missing or incorrect.

**SPLBPD-958, "Differential pair %s cannot be created because the negative pin definition is missing or incorrect. Select the two pins and choose Create Differential Pair from the shortcut (RMB) menu. If you do not want to create the differential pair, remove the DIFF\_PAIR\_PINS\_POS property from the positive pin in chips.prt.",E**

This error message appears when loading a part if a positive pin definition is found and the corresponding negative pin definition is missing or incorrect.

**SPLBUI-2, "Physical pin number %s already exists.",E**

This error message is generated in the following conditions:

- In the Global Pin Map – Edit option, if a new pin number is added by inserting a row and the pin number already exists.
- If a new pin number is added using the Add Physical Pins options manually or directly by inserting a row in the Physical Pin grid and the new pin number already exists.

**SPLBUI-3, "Physical pin number %s already mapped to pin %. Specify a unique physical pin number.",E**

This error is generated in the following conditions:

- If a duplicate pin number is added by editing the mapping column in the Global Pins grid.
- If a pin number is already mapped to a global pin and an attempt is made to map it again to the same global pin.
- If a duplicate pin number is added by using *Footprint – Add Physical Pins Manually* or by editing any of the logical or physical pin grids or an attempt is made to map an already mapped physical pin to a different pin name (logical or global).

**SPLBUI-5, "At a time, only either logical pins or global signals can be selected",E**

This error is generated if pins in both logical pin and global pin grids are selected simultaneously. You can select pins from either the logical pin grid or the global pin grid at one time.

**SPLBUI-6, "At a time, only one global signal can be mapped.",E**

This error is generated if an attempt is made to map two or more global pins in the global pin grid. You can map only one global pin at a time. To map multiple global pins simultaneously, see [Global Pin Map – Edit](#) on page 133.

**SPLBUI-7, "Select unmapped pin(s).",E**

This error is generated if you click on *Map* without selecting any pin in any of the three grids in the Cell Editor.

**SPLBUI-8, "You have selected pin name(s) only. Select an equal number of physical pin number(s) and then click Map.",E**

This error is generated if only a logical or global pin is selected and no physical pin is selected and an attempt to map is made.

**SPLBUI-9, "No mapped pins selected for unmapping. Select one or more mapped physical pins in the Logical Pins grid.",E**

This error is generated if an *Unmap* action is attempted on already unmapped pins.

**SPLBUI-10, "Select logical pin(s) to move to the Global Pins grid.",E**

This error is generated if an attempt is made to move a logical pin to the *Global Pins* grid without selecting any logical pin in the *Logical Pins* grid.

**SPLBUI-11, "Select global pin(s) to move to the Logical Pins grid.",E**

This error is generated if an attempt is made to move a global pin to the *Logical Pins* grid without selecting any global pin from the *Global Pins* grid.

**SPLBUI-13, "Mapped pins cannot be deleted. First, unmap the pins.",E**

This error is generated if you try to delete a mapped pin. You must unmap the pin before deleting it.

**SPLBUI-14, "Duplicate name %s specified. Specify unique name(s).",E**

This error is generated in the following conditions:

- Default properties are given the same name on the Select Destination page in FPGA import.
- Two properties are given the same name when adding using the *Properties – Add* option.
- A property name is renamed and has the same name as an existing property.
- Two pins are renamed to a same name.

**SPLBUI-15, "Cell %s already open",E**

This error is generated if an attempt is made to open an already open cell.

**SPLBUI-16, "Project has been modified outside of Part Developer. Would you like it to be reloaded?",Q**

This query is generated whenever a project that is currently loaded in Part Developer is modified outside of Part Developer, such as using Project Manager.

**SPLBUI-17, "Symbol %s of %s has been modified outside of Part Developer. Would you like it to be reloaded?",Q**

This query is generated whenever a symbol is modified outside of Part Developer, such as in Design Entry HDL, and the cell to which the symbol belongs is currently loaded in Part Developer.

**SPLBUI-18, "PTF %s of %s has been modified outside of Part Developer. Would you like it to be reloaded?",Q**

This query is generated whenever a cell-level PTF is modified outside of Part Developer, such as in Part Table Editor, and the cell to which the PTF belongs is currently loaded in Part Developer.

**SPLBUI-19, "The logical part cannot be deleted because it is the only logical part.",E**

This error is generated if an attempt is made to delete the only existing logical part in a package. For more information about logical and physical parts, see *Packager-XL Reference*.

**SPLBUI-20, "Logical part cannot be renamed as physical part %s already exists.",E**

This error is generated if a new package is added and a part is renamed with the same name as that of an existing part in some other package.

**SPLBUI-21, "A vector pin with the same basename %s already exists. Specify a unique name.",E**

This error is generated if a scalar pin is added when a vector pin with the same name is already present.

**SPLBUI-22, "A scalar pin with the same basename %s already exists. Specify a unique name.",E**

This error is generated if a vector pin is added when a scalar pin with the same name is already present.

**SPLBUI-23, "The default package already exists. To add a new package, choose New from the pop-up menu.",E**

This error is generated if the default physical part for the logical part already exists and the *Add Default* option is selected.

**SPLBUI-24, "Save changes to cell %s of library %s?",Q**

This query is generated if an attempt is made to close a cell without saving it.

**SPLBUI-25, "Name cannot be empty. Specify a unique name.",E**

This error is generated in the following conditions:

- A *New* or *Rename* operation is attempted for a logical part in the Package Editor and the name is left empty.

- If the *Rename* option is selected for any symbol, part table file, VHDL wrapper and so on in the Cell Editor and the name is left empty.
- If *New* or *Rename* is attempted for a physical part in the Package Editor and the name is left empty.

**SPLBUI-26, "The part does not have any pin to distribute. Add the required pins and functions and then click Distribute Pins.",E**

This error is generated if there are no pins in the Logical Pin grid and an attempt is made to distribute pins.

**SPLBUI-27, "The part does not have functions. Add the required functions and then click Distribute Pins.",E**

This error is generated if there are no functions in the *Edit Functions* dialog box and *Distribute Pins* is clicked. For more information, see [Entering Package Information](#) on page 123, [Edit Functions](#) on page 408 and [Distribute Pins](#) on page 409.

**SPLBUI-28, "The selected pins cannot be moved to the Global Pins grid. Select pins of type POWER, GROUND, or NC.",E**

This error is generated when an attempt is made to move a logical pin from the *Logical Pins* grid to the *Global Pins* grid and it is not of type POWER, GROUND, or NC.

**SPLBUI-29, "Pins are not distributed across all %s slots. Delete the empty slot(s) or distribute the pins across all the slots.",E**

This error is generated if any of the slots is left empty when distributing pins. For more information, see [Entering Package Information](#) on page 123, [Edit Functions](#) on page 408 and [Distribute Pins](#) on page 409.

**SPLBUI-31, "The selected objects have been synchronized.",I**

This message is displayed when interface comparison is done on two objects and the synchronization is done on them.

**SPLBUI-32, "Some pins have been added in the package as a result of synchronization. Specify the pin numbers for the added pins.",W**

This warning is generated when interface comparison is done and some logical pins are added by the synchronization tool. For more information, see [Chapter 14, “Interface Comparator.”](#)

**SPLBUI-33, "Physical part %s already exists. Specify a unique part name.",W**

This warning is generated if a physical part is added with the same name as an already existing part name.

**SPLBUI-34, "The physical part cannot be deleted because it is the only physical part.",E**

This error is generated if only one physical part exists in the cell and a delete is attempted on it.

**SPLBUI-35, "Pins without names cannot be moved to the Global Pins grid.",E**

This error is generated if an attempt is made to move a pin without a name from the *Logical Pins* grid to the *Global Pins* grid.

**SPLBUI-36, "Pins without names cannot be moved to the Logical Pins grid. Specify a pin name for each pin to be moved.",E**

This error is generated if an attempt is made to move a pin without a name from the *Global Pins* grid to the *Logical Pins* grid.

**SPLBUI-37, "Specify a pin name.",E**

This error is generated if an attempt is made to add a scalar, vector, or sizeable pin without specifying a pin name in the Add Pin dialog box.

**SPLBUI-38, "Logical part %s already exists. Specify a unique part name.",E**

This error message is generated if a logical part is created with the same name as an existing logical part.

**SPLBUI-39, "The pin(s) cannot be moved because they have errors. Correct the errors and then retry.",E**

This error is generated if an attempt is made to move a pin from the Logical Pins grid to the Global Pins grid and the pin has some errors.

**SPLBUI-40, "The pin(s) cannot be moved because they have errors. Correct the errors and then retry.",E**

This error is generated if an attempt is made to move a pin from the *Global Pins* grid to the *Logical Pins* grid and the pin has some errors.

**SPLBUI-41, "A vector pin must have values for both LSB and MSB. Specify the missing value or delete the existing value to add the pin as scalar.",E**

This error is generated if only the *LSB* value or the *MSB* value has been specified when creating a vector pin. You need to specify both *LSB* and *MSB* values for creating a vector pin.

**SPLBUI-42, "LSB is smaller than MSB. Reenter correct MSB and LSB values (for example, for vector pin A<4..1>, MSB is 4 and LSB is 1).",E**

This error is generated when the value specified for LSB is greater than the value specified for MSB.

**SPLBUI-43, "Specify pin number(s) to add.",E**

This error is generated if no pin number is specified in the Add Physical Pin Numbers dialog box.

**SPLBUI-44, "Specify row and column labels before clicking the Create button.",E**

This error is generated if the *Grid* option is used in the Add Physical Pin Numbers dialog box and row label and column label values are not specified before clicking *Create*.

**SPLBUI-45, "No row labels specified. Enter a range of values in the Row Labels field before clicking the Create button.",E**

This error is generated if the *Grid* option is used in the Add Physical Pin Numbers dialog box and row label values are not specified before clicking *Create*.

**SPLBUI-46, "No column labels specified. Enter a range of values in the Col Labels field before clicking the Create button.",E**

This error is generated if the *Grid* option is used in the Add Physical Pin Numbers dialog box and column label values are not specified before clicking *Create*.

**SPLBUI-47, "A physical pin number cannot be 0. Specify non-zero natural numbers as pin number(s).",E**

This error is generated in the following conditions:

- In the Add Physical Pins Manually dialog box, the pin number is set to 0, such as 0--5 or 0-5.
- A pin number is set to 0 in the *Global Pins* grid for a new or existing pin or a pin number is specified as 0 in the *Global Pin Map – Edit* option.

In Cadence flows, pin number 0 is reserved and cannot be used.

**SPLBUI-48, "Select all bits of the vector pin to move the pin to the Global Pins grid.",E**

This error is generated if a vector pin is selected to be moved down to the *Global Pins* grid without selecting all of its bits.

**SPLBUI-49, "The selected column is not a property column. Select a property column (for example, PIN\_DELAY) and make sure that the property is associated with pins(s).",E**

This error is generated if *Properties – Attributes* is selected without selecting any property column in the *Symbol Pins* grid.

**SPLBUI-50, "The property is not associated with any pins. To view the property attributes, first associate the property with a pin by assigning a property value.",E**

This error is generated if *Properties – Attributes* is selected and the property selected in the *Symbol Pins* grid is not present in any pin.

**SPLBUI-51, "Cannot map an erroneous pin number to a logical pin",E**

This error is generated if an attempt is made to map an erroneous physical pin to a logical pin.

**SPLBUI-52, "Cannot map pin number 0 to a logical pin because the pin number is invalid. Assign a valid pin number and make sure it is unique.",E**

This error is generated if an attempt is made to map a pin numbered zero to a logical pin. Pin number 0 is reserved and should not be used.

**SPLBUI-53, "The numbers of logical pins and physical pins selected for mapping are not equal. Select an equal number of logical and physical pins to map.",E**

This error is generated when an unequal number of pins or ports and logical pins are selected and a mapping is attempted.

**SPLBUI-54, "The selected pin(s) are already mapped. To map pins, select pins that are not mapped.",E**

This error is generated if an attempt is made to map pins that are already mapped. You need to unmap the pins before you can map them to different pins.

**SPLBUI-55, "Logical pin(s) having a mode different from the mode of model port(s) ignored for mapping",E**

This error is generated if in a VHDL model or wrapper file or a Verilog model or wrapper file under the mapping tab, a logical pin is selected with a mode different from the mode of the selected model port and *Map* or *Automap* is clicked.

**SPLBUI-56, "Select the logical pin(s) to map.",E**

This error is generated if under the mapping tab in a VHDL or Verilog model or wrapper file, no logical pin is selected but some model port(s) is/are selected and the *Map* button is clicked.

**SPLBUI-57, "Select the model port(s) to map.",E**

This error is generated if under the mapping tab in a VHDL or Verilog model or wrapper file, no model port is selected but some logical pin(s) is/are selected and the *Map* button is clicked.

**SPLBUI-58, "Select the logical pin(s) to unmap.",E**

This error is generated if under the mapping tab in a VHDL or Verilog model or wrapper file, no logical pin is selected and the *Unmap* button is clicked.

**SPLBUI-59, "Logical pin(s) having a type different from the type of model port(s) ignored for mapping",E**

This error is generated if in a VHDL model or wrapper file under the mapping tab, a logical pin is selected with mode different than the mode of the selected model port and Map or Automap is clicked.

**SPLBUI-60, "Selected packages %s do not meet the conditions for clubbing. See Help for details.",E**

This error is generated if the selected packages have different numbers of pins when adding or modifying packages in a VHDL or Verilog map file.

**SPLBUI-61, "Model port(s) that are already mapped ignored for mapping",E**

This error is generated if under the mapping tab in a VHDL or Verilog model, an attempt is made to map model ports and logical pins that are already mapped.

**SPLBUI-62, "Select a jedec type.",E**

This error message is selected in the following conditions:

- Select Alt Symbol is used and nothing is selected before clicking OK.
- Browse for Jedec Type is used and nothing is selected before clicking OK.
- Select using Ptf for footprint is used and nothing is selected before clicking OK.

**SPLBUI-63, " Sizeable pins %s in symbol %s do not have the same width. Change the common pins from sized to scalar.",E**

This error is generated when pins on the selected sizeable symbol have different size widths. For more information, see [Creating Sizeable and HAS\\_FIXED\\_SIZE Symbols](#) on page 138.

**SPLBUI-64, "Select an equal number of pins and text items to associate.",E**

This error is generated when an unequal number of pins and text items are selected for association. For more information, see [Map Pin Name and Text](#) on page 421.

**SPLBUI-65, "The model selected has module name same as that in the entity. This is not allowed for Verilog.",E**

This error is generated if the selected model has the same name as that in the entity. For more information, see [Chapter 10, “Creating Verilog Wrappers and Map Files.”](#)

**SPLBUI-66, "Select annotated parameter(s).",E**

This error is generated when no annotated parameters are selected. For more information, see [Chapter 10, “Creating Verilog Wrappers and Map Files.”](#)

**SPLBUI-67, "Model port(s) that have no mode ignored for mapping",E**

This error is generated when model ports with a mode type are ignored for mapping. For more information, see [Chapter 9, “Working with VHDL Wrappers and Map Files,”](#) and [Chapter 10, “Creating Verilog Wrappers and Map Files.”](#)

**SPLBUI-68, "Unequal number of pins selected for mapping (logical pins selected = %s, physical pins selected = %s)",E**

This error is generated if the number of pins selected in the Logical Pins grid is not equal to the number of pins selected in the Physical Pins grid.

**SPLBUI-69, "Part table file %s already exists. Specify a unique filename.",E**

This error is generated if a part table file is renamed to an already existing part table file.

**SPLBUI-70, "One or more properties have null values. They might get lost if the cell tree selection changes.",W**

This warning is generated if a property is added by specifying only its name and no value.

**SPLBUI-71, "To and From values can be either both letters or both numbers.",E**

This error is generated if both *To* and *From* values are not either letters or numbers. You cannot mix letters and numbers in the *To* and *From* fields.

**SPLBUI-72, "Package %s already exists. Specify a unique package name.",E**

This error is generated if the package already exists.

**SPLBUI-73, "Symbol %s already exists. Specify a unique symbol name.",E**

This error is generated if the symbol already exists.

**SPLBUI-74, "Mapview %s already exists. Specify a unique mapview name.",E**

This error is generated if the mapview already exists.

**SPLBUI-75, "Wrapper %s already exists. Specify a unique wrapper name.",E**

This error is generated if the wrapper already exists.

**SPLBUI-76, "Wrapper model %s already exists in wrapper %. Specify a unique model name.",E**

This error is generated if a model is copied and pasted and the copied model already exists in the wrapper.

**SPLBUI-77, "Mapview primitive %s already exists in mapview %. Specify a unique primitive name.",E**

This error is generated if in VHDL or Verilog map files, a mapview primitive is copied and pasted and the primitive already exists in the map file.

**SPLBUI-78, "Mapview model %s already exists in primitive %s of mapview %. Specify a unique model name.",E**

This error is generated in the following cases:

- In VHDL or Verilog map files, a model is added to one of the mapview primitives and the model already exists in that primitive.
- In VHDL or Verilog map files, a model in one of the mapview primitives is copied and pasted into another primitive and the model already exists in that primitive.

**SPLBUI-79, "The selected pin(s) are already associated. To change the association, first click the Unassociate button. To associate selected pins with new pin text, select pin text from the Unassociated Text grid and click the Associate button.",E**

This error is generated if the selected pins already have pin text associated with them. For more information, see [Map Pin Name and Text](#) on page 421.

**SPLBUI-80, "%s already exists. Do you want to overwrite?",Q**

This query is generated if the cell name specified to save already exists in the selected library.

**SPLBUI-81, "Could not overwrite cell %s as it is already opened. To address the sharing violation and overwrite the cell, close it first.",E**

This error is generated if the cell name specified in Save As already exists and is currently open in Part Developer. You need to close the cell and then attempt to save the part again.

**SPLBUI-82, "Cannot paste already existing mapview primitive(s) in mapview %s",E**

This error is generated if you try to paste a mapview and that mapview already exists in the cell.

**SPLBUI-83, "Pin name %s already exists in the package. Specify a unique pin name.",E**

This error is generated if a duplicate pin name is specified. Pin names must be unique.

**SPLBUI-84, "Pin name %s already exists. Specify a unique pin name.",E**

This error is generated if a duplicate pin name is specified. Pin names must be unique.

**SPLBUI-85, "Bit-specific properties might get lost on collapsing. Do you want to proceed?",Q**

This query is generated when you try to collapse the bits of a vector pin where the bits have different property values. The values will get lost if you collapse the bits.

**SPLBUI-86, "Do you really want to delete %s?",Q**

This query is generated when you attempt to delete a package, symbol, Verilog wrapper, or part table file.

**SPLBUI-87, "Select a package and then click OK.",E**

This error is generated if the *Modify Package* option is selected in the Cell Editor for a VHDL or Verilog model and no package is selected for modification.

**SPLBUI-88, "Package %s cannot be selected as it does not exist in the cell.",E**

This error message is generated if the *Modify Package* option is selected in the Cell Editor for a VHDL or Verilog model and the selected package to be modified does not exist.

**SPLBUI-90, "Unable to read %s ",E**

This error is generated in the following cases:

- Either the PTF or the chips.prt file cannot be read or does not exist.
- The View Log File option is chosen after Verify and the cp.msg file is not found in the checkplus directory. Check your installation and re-install if necessary.

**SPLBUI-91, "Select a PTF file.",E**

This error is generated if the PTF Editor is not loaded in the cell and the *View Part Table* option is selected.

**SPLBUI-92, "Add a comment.",E**

This error message is generated if the *Change Log* option is selected on the *Revision* tab and then an attempt is made to close the Add User Comments dialog box without adding any comments.

**SPLBUI-93, "Cadence Help cannot be started because of a problem with the help request received from the Cadence application you are using. Contact Cadence Customer Support if the problem persists.",E**

This error is generated when the handle returned by the Cadence Help API call is invalid. If the problem persists, report this problem by contacting Cadence Customer Support to file a Cadence Change Request (CCR) with the following settings:

Family=Framework

Product=CDNSHELP

Provide as much detail as possible in the CCR description.

**SPLBUI-94, "Paste of the item cut/copied not allowed",E**

This error is generated when you attempt to copy/paste a header node, such as Symbol or Package. You can copy/paste specific packages and symbols.

**SPLBUI-95, "Verilog wrapper %s already has one model instantiation.",E**

This error is generated when a model is copied and pasted in the Verilog wrapper and the copied model already exists in the wrapper.

**SPLBUI-96, "Pin name %s with the same base name and hyphen prefix already exists.",E**

This error is generated if a low-asserted pin is already present with a hyphen prefix and a new low-asserted pin with the same name is added using the *Add Pins* option in the Package Editor. By default, Part Developer treats all pins with a hyphen prefix as low-asserted pins.

**SPLBUI-97, "Pin name %s with the same base name and low-asserted pin already exists.", E**

This error is generated if a low-asserted pin is already present with the same base name and the low-assertion symbol as defined in the *Setup* option and a new low-asserted pin with the same base name and a hyphen prefix is added using the *Pins – Add* option in the Package Editor.

**SPLBUI-98, "Options to treat aliases as different primitives or individual cells is not available from here. Use Library Flows for this.", E**

This error is generated when you select the option to treat aliases as different primitives or individual cells in Capture export. This option is available only through the Library Flows option in the Project Manager tool.

**SPLBUI-100, "A %s pin with base name %s already exists in the cell. Cannot add %s pin with the same base name.",E**

This error is generated when the part already has a pin with the same base name as you have specified. Specify a unique base name.

**SPLBUI-101, "A %s pin with the same base name and hyphen prefix %s already exists in the cell. Cannot add %s pin with the same base name and low assertion.",E**

This error is generated if a low-asserted pin with the suffix (as specified in Setup) exists and a pin with a hyphen prefix and the same base name is added by editing the *Logical Pins* grid. For more information, see [Setting Up Defaults for Low-Asserted Pins and Split Parts](#) on page 83.

**SPLBUI-102, "A %s pin with the same base name and low assertion %s already exists in the cell. Cannot add %s pin with the same base name and hyphen prefix.",E**

This error is generated if a low-asserted pin with hyphen as the prefix exists and a pin with the suffix specified in Setup and the same base name is added by editing the *Logical Pins* grid. For more information, see [Setting Up Defaults for Low-Asserted Pins and Split Parts](#) on page 83.

**SPLBUI-103, "Pin name %s with the same base name and hyphen prefix already exists in the package.",E**

This error is generated if a low-asserted pin is present with the same name and a hyphen and you try to add a new low-asserted pin with the same name by editing pins in the *Logical Pins* grid. For more information, see [Setting Up Defaults for Low-Asserted Pins and Split Parts](#) on page 83.

**SPLBUI-104, "Pin name %s with the same base name and low assertion already exists in the package.",E**

This error is generated if a low-asserted pin exists with the low-assertion symbol as defined in the *Setup* option and you try to add a new low-asserted pin with the same base name and hyphen by editing pin names in the *Logical Pins* grid. For more information, see [Setting Up Defaults for Low-Asserted Pins and Split Parts](#) on page 83.

**SPLBUI-108, "Mapping will be lost if you delete the slot. Do you want to map the pins in this slot to other slots?",Q**

This message is given when a slot in which some pins are present is deleted using the *Delete* button in the Edit Functions dialog box.

**SPLBUI-109, "All mapped or unmapped pins not present in the footprint will be deleted. Do you want to proceed?",Q**

This query is generated when the footprint selection option is used and a footprint is selected. When you select a footprint, all the existing physical pins that are not present in the selected footprint are deleted from the package.

**SPLBUI-110, "All the bits of pin(s) %s must be present in one function only.",E**

This error is generated if the *Pin on one symbol only* check box is selected in the Distribute Pins dialog box and bits of vector pins are already in a different function. This message is displayed as each bit should be on a single symbol.

**SPLBUI-111, "Pin(s) %s must be present in one function only.",E**

This error is generated if the *Pin on one symbol only* check box is selected in the Distribute Pins dialog box and some scalar pins that are present in the package and not selected for a function/slot.

**SPLBUI-112, "Physical pins %s are present in the chips but not in footprint %s.",E**

This error is generated if the *Select & Verify With Footprint* option is used and some physical pins are present in `chips.prt` but not in the footprint.

**SPLBUI-113, "Physical pins %s are present in footprint %s but not in the chips.",E**

This error is generated if the *Select & Verify With Footprint* option is used and some physical pins are present in the footprint but not in the `chips.prt` file.

**SPLBUI-115, "Choose Pins - Global Delete to delete pins that are present in any package or symbol.",E**

This error is generated if the *Delete Selected Rows* option is chosen in the Add Pin dialog box on pins that are already present in one or more packages. You can use the *Delete Selected Rows* option to only delete those pins that are not present in any package.

**SPLBUI-116, "Cut operation works only for the grids.",E**

This error is generated if anything other than a cell, row, or column is selected from a grid in the Cell Editor and the *Cut* option is chosen from the *Edit* menu.

**SPLBUI-118, "Pin names %s are duplicate. Specify unique pin names.",E**

This error is generated if the specified logical pin already exists.

**SPLBUI-119, "Cell %s of library %s cannot be exported to ViewLogic because it does not have any symbol. Generate symbols and then export the part.",E**

This error is generated if the cell from which packages or symbols are to be exported does not contain any symbols.

**SPLBUI-120, "You have selected both a package and a symbol. Select either a package or a symbol.",E**

This error message is generated if you select both a package and a symbol for export. You can select either a package or a symbol.

**SPLBUI-121, "Select a package or symbol to export.",E**

This error is generated if neither a package nor a symbol is selected for export.

**SPLBUI-122, "Multiple packages can be selected only if they differ due to the presence of visible and invisible power pins.",E**

This error is generated when multiple packages are selected in Viewlogic export and they do not differ only in visible and invisible power pins.

**SPLBUI-123, "Import cannot proceed because standard component %s of library %s is already open. Close the part and then reimport.",E**

This error is generated if the standard component specified in FPGA import is already open and an import is attempted.

**SPLBUI-124, "The standard component and the cell to be created after import cannot be same.",E**

This error is generated if the standard component specified in FPGA import and the cell to be created after import are same.

**SPLBUI-125, "Cell %s already created in library %s. Select a different name for cell",E**

This error is generated if in the import options, a cell has been created but not saved and an attempt is made to import another cell with the same name.

**SPLBUI-127, "Directory does not exist",E**

This error is generated if the specified directory does not exist.

**SPLBUI-128, "Directory does not have write permission",E**

This error is generated if the specified directory does not have write permission.

**SPLBUI-129, "%s is not a directory path.",E**

This error is generated if the specified path is a file path and not a directory path.

**SPLBUI-130, "Select only one Allegro footprint for importing.",E**

This error is generated if more than one footprint are selected in the Footprints grid while importing a footprint. Make sure you select only one Allegro footprint.

**SPLBUI-151, "No sizeable pin exists on the symbol. Add a sizeable pin through the Add Pin dialog box in the Symbol Editor and then click Set Size.",E**

This error is generated if *Set Size* is clicked on a symbol that has no sizeable pins. For more information, see [Creating Sizeable and HAS\\_FIXED\\_SIZE Symbols](#) on page 138.

**SPLBUI-200, "Either the first or last non-empty row can be set as the header.",E**

This error is generated if an attempt is made to set a row other than the first or last non-empty row as a header row by using *Set Row as Header* in pin grid import. For more information, see [Chapter 6, “Creating Parts from PDFs.”](#)

**SPLBUI-201, "Either the first or last non-empty column can be set as the header.",E**

This error is generated if an attempt is made to set a column other than the first or last non-empty column as a header column by using *Set Col as Header* in pin grid import. For more information, see [Chapter 6, “Creating Parts from PDFs.”](#)

**SPLBUI-205, “Some Cadence online documentation is missing or cannot be opened due to an error (error code %s). For troubleshooting, see the Cadence Help documentation at the <install\_dir>/doc/cdnshelp location.”,E**

This error is generated when the Cadence Help documentation system fails to launch. The exact reason can be ascertained using the error code. For details and troubleshooting information, see the Cadence Help User Guide at `<install_dir>/doc/cdnshelp` location.

**SPLBUI-300, "Pin name column(s) not set",E**

This error is generated if any operation such as Merge, Next on PDF pin table is attempted without setting any column as the Pin Name column. For more information, see [Chapter 6, “Creating Parts from PDFs.”](#)

**SPLBUI-301, "Number of columns set for pin number should be equal to the number of pin name columns.",E**

This error is generated if the number of Pin Name columns is not equal to the number of Pin Number columns when pin number columns is not equal to zero.

**SPLBUI-302, "Number of columns set for pin type should be equal to the number of pin name columns.",E**

This error is generated if the number of Pin Type columns is not equal to the number of Pin Name columns when the number of Pin Type columns is not equal to zero in the pin table.

**SPLBUI-303, "At a time, only one row or column can be filled.",E**

This error is generated if multiple rows or columns are selected and the *Fill Cells* option is used. The *Fill Cells* option works on one row or column at a time.

**SPLBUI-304, "Both ends of the selection must be non-empty to fill automatically.",E**

This error is generated if both the ends of a selection do not contain range information to fill the cells in between and the *Fill Cells* option is used.

**SPLBUI-305, "Use the Table/Formatted Text Select Tool option to select data from Adobe Acrobat 5.0 & above (not available in Acrobat Reader)",E**

This error is generated if data is pasted on the grid in a single column. Part Developer assumes this as incorrect copy/paste from Adobe Acrobat. You can ignore this error if the data is pasted correctly. For more information, see [Chapter 6, “Creating Parts from PDFs.”](#)

**SPLBUI-306, "Select rows or columns as pin names or pin numbers.",E**

This error is generated if no selection is made for the pin name or pin number area.

**SPLBUI-307, "Multiple logical pins %s cannot be mapped to a single physical pin. Convert the bundled pin to either scalar or vector bits.",E**

**SPLBUI-308, "A shape with the same name already exists in the directory. Select a different name for the shape.",E**

**SPLBUI-309, "Close the Shape Editor first and then close the project.",E**

**SPLBUI-312, "Pad file %s used in footprint %s for physical pins %s is not available in the path.",E**

**SPLBUI-313, "Flash file %s used in %s is not available in the path.",E**

**SPLBUI-315, "Shape %s will not be placed on the symbol outline because it is shorter than the stub length defined in symbol pin setup. Open the shape in the Shape Editor and increase its length. Alternatively, you can modify the height and width specified in the Maximum Size (In Grid Units) option in shape setup.",W**

This warning appears when you try to apply a pin shape that is shorter than the stub length specified in symbol pin shape. If you apply such a shape to a symbol pin, the pin will not be placed on the symbol outline. To solve the problem, you can increase the length of the shape in the Shape Editor to the number of grid units set as the stub length.

Alternatively, you can retain the shape length and follow one of these methods:

- Stretch the shape when it is attached to a symbol pin

Modifying the default height and width in the *Maximum Size (In Grid Units)* option in shape setup enables you to define how pin shapes are to be stretched or shrunk when they are attached to symbol pins.

- Reduce the stub length in symbol pin setup

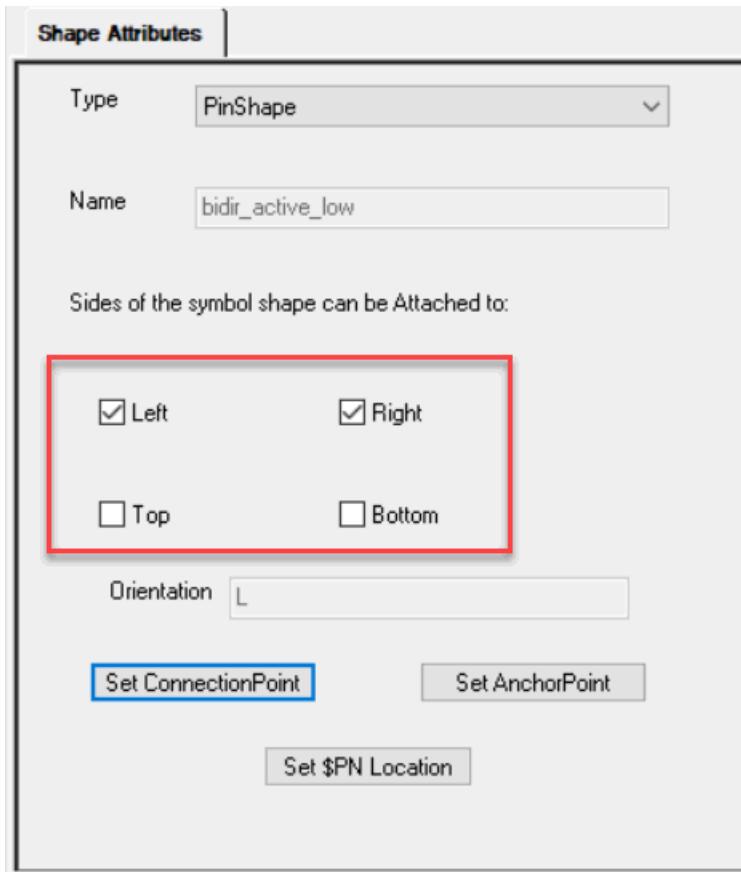
**SPLBUI-316, "Shape %s should not be applied at the current symbol location. Check the shape attributes and attach the shape to the side of the symbol to which it should be attached. Alternatively, you can modify the shape attributes.",W**

The attributes of a shape include the specification regarding the sides of the symbol to which the shape should be attached. When you try to apply the shape to a symbol side to which the

## Part Developer User Guide

### Errors and Warnings

shape is not designed to be attached, Part Developer displays this warning. Ignoring the warning might cause distortion of the symbol.



---

# Shortcut Menu Options

---

Several of the part creation tasks can be done through the shortcut menu options provided by Part Developer. This chapter covers the following shortcut menu options:

- From each node of the cell tree
- From the Logical and Physical Pins grids
- From the properties grids

## Shortcut Menu Options from Cell Tree Nodes

### Packages

The following shortcut menu option is available from the Packages node:

Menu Option	Description
New	Creates a new package

### Package Names

The following shortcut menu options are available from the package names:

Menu Option	Description
Copy	Copies the selected package into the clipboard.

## Part Developer User Guide

### Shortcut Menu Options

---

Menu Option	Description
Paste	Pastes the information in the clipboard into the Cell tree.
Delete	Deletes the selected package
Generate Symbol(s)	Generates symbols from the selected package. For details, see <a href="#">Direct Generation from a Package</a> on page 135 for details.
Interface Comparison	Compares the pin lists of two interfaces, such as a package and a symbol, identify the differences and then update the pin list of one of the objects to match the other, or both with respect to each other. For more information, see the <i>Interface Comparator</i> chapter.

## Symbols

The following shortcut menu option is available from the Symbols node:

Menu Option	Description
New	Creates a new symbol

## Symbol Names (sym\_n)

The following shortcut menu options are available from the symbol names:

Menu Option	Description
Copy	Copies the selected symbol into the clipboard.

## Part Developer User Guide

### Shortcut Menu Options

---

Menu Option	Description
Paste	Pastes the information in the clipboard into the Cell tree.
Delete	Deletes the selected symbol
Rename	Renames the selected symbol
Edit	Launches Allegro Design Entry HDL on the selected symbol
Generate Package	Generates a package
Add FunctionGroup to a Package	Add the function group on which the symbol is based to a package. The package to which the function group can be associated should not be associated with any symbol.
Interface Comparison	Compares the pin lists of two interfaces, such as a package and a symbol, identify the differences and then update the pin list of one of the objects to match the other, or both with respect to each other. For more information, see the <i>Interface Comparator</i> chapter.

## Part Table Files

The following shortcut menu option is available from the Part Table Files node:

Menu Option	Description
New	Creates a new part table file

## Part Table File Names (<name>.ptf)

The following shortcut menu options are available from the part table file names:

## Part Developer User Guide

### Shortcut Menu Options

---

<b>Menu Option</b>	<b>Description</b>
Copy	Copies the selected part table file into the clipboard.
Paste	Pastes the information in the clipboard into the Cell tree.
Delete	Deletes the selected part table file
Rename	Renames the selected part table file
Edit	Launches Part Table File editor on the selected part table file
View	Launches the part table files in a texteditor

## VHDL/Verilog MapFiles

The following shortcut menu option is available from the VHDL/Verilog MapFiles node:

<b>Menu Option</b>	<b>Description</b>
New	Creates a new VHDL/Verilog mapfile

## VHDL/Verilog mapfile names

The following shortcut menu options are available from the VHDL mapfile names:

<b>Menu Option</b>	<b>Description</b>
Copy	Copies the selected VHDL/Verilog map file into the clipboard.
Paste	Pastes the information in the clipboard into the Cell tree.

## Part Developer User Guide

### Shortcut Menu Options

---

Menu Option	Description
Delete	Deletes the selected VHDL/Verilog map file
Rename	Renames the selected VHDL/Verilog map file.
Select package	Displays the list of packages from which pin information is got to create the map file. On selecting a package, a new primitive entry is added to the map file.

### Primitive Names in VHDL/Verilog Mapfile Names

The following shortcut menu options are available from the primitive names in VHDL/Verilog mapfile names:

Menu Option	Description
Copy	Copies the selected primitive into the clipboard.
Paste	Pastes the information in the clipboard into the Cell tree.
Delete	Deletes the selected primitive from the map file
Modify package	Displays the list of packages from which pin information is got to create the map file. On selecting a package, the primitive name and the pin list is updated.
Modify properties	Modifies the default model and the UPPER_CASE property.
Add Model	Adds a model to the selected primitive

## **VHDL/Verilog Wrappers**

The following shortcut menu option is available from the VHDL/Verilog Wrappers node:

Menu Option	Description
New	Creates a new VHDL/Verilog wrapper

## **VHDL/Verilog map file names**

The following shortcut menu options are available from the VHDL wrapper names:

Menu Option	Description
Copy	Copies the selected VHDL/Verilog map file into the clipboard.
Paste	Pastes the information in the clipboard into the Cell tree.
Delete	Deletes the selected VHDL/Verilog map file
Rename	Renames the selected VHDL/Verilog map file.
Add Model	Adds a model to the wrapper

## **Model Names in VHDL/Verilog Wrappers**

The following shortcut menu options are available from the model names in VHDL/Verilog wrappers:

Menu Option	Description
Copy	Copies the selected primitive into the clipboard.

Menu Option	Description
Paste	Pastes the information in the clipboard into the Cell tree.
Delete	Deletes the selected primitive from the map file
Modify Model	Modifies the model information in the wrapper

## Menu Options in Logical/Physical/Global Pins and Properties Tables

The following menu options are available from the pins and properties table across the different editors.

Menu Option	Description
Modify Values	Displays the Modify Column Values dialog box through which you can modify the values of the selected columns.
Delete Selected Rows	Deletes the selected rows from the table
Filter Rows	Displays the Filter Rows dialog box through which you can specify the filter values. Only those rows that match the specified values are displayed after the filter is applied.
<b>Note:</b> To see all the rows, select <i>Unhide all Rows</i>	
Select All	Selects the complete table
Deselect All	Deselects the selection
Invert Selection	Deselects all the rows and columns that were selected and selects the rows and columns that were not selected
Hide Load Cols	Hides the <i>Input Load</i> and <i>Output Load</i> columns in the Package Editor
Hide Mapped Pins	Hides the pins that are mapped.
Hide Selected Columns	Hides the selected columns from the table

Menu Option	Description
Hide Selected Rows	Hides the selected rows from the table
Unhide all Rows	Shows all the hidden rows
Unhide all Cols	Shows all the hidden columns
Insert Row After	Inserts a row below the selected row
Insert Row Before	Inserts a row above the selected row
Expand	Expands the bits of a vector pin
Collapse	Collapses the bits of a vector pin
Select All Bits	Selects all bits of a vector pin
Reverse Order	Reverses the bit order in the symbol

## Grid Usability Tips

Part Developer provides several features that enable you to speed up part creation and modification. This section provides a list of useful features and how to use them.

### Modify Values

#### ***Available from:***

Right-click on any pin grid

The *Modify Values* option enables you to change the values in the selected columns in a context-sensitive way. For more information, see [Modify Column Values](#) on page 447.

You can also use the Modify Values feature directly when working with grid contents. For example, if you want to modify the pin type for a set of pins, you can make the change for the first pin and then select the rest of the pins.

## Filter Rows

**Available from:**

Right-click on the pin grids

The *Filter Rows* option enables you to view only those rows that match the filter values. For more information, see [Creating a Filter](#) on page 174 and [Removing a Filter](#) on page 175.

## Creating Vector Pins from Scalar Pins

**Available from:**

Right-click on the Logical Pins grid

Using a single-click action, you can convert scalar pins to bits of a vector pin. For more information, see [Converting Scalar Pins to Vector](#) on page 167

## Set Origin

**Available from:**

Symbol Editor

The *Set Origin* option enables you to set the origin of a symbol. For more information, see [Set Origin](#) on page 62.

## Pass-Through and Bubble Pins Creation

**Available from:**

Symbol Editor

The Symbol Editor enables you to create pass-through and bubble pins. For more information, see [Creating Sizeable and HAS\\_FIXED\\_SIZE Symbols](#) on page 138.

## **Adding PIN\_GROUP and other pin level properties**

### ***Available From:***

Setup, Symbol and Package Editors

You can add the pin-level properties from within Setup, Symbol and Package Editors. For more information, see the following:

- [Setting Up Package Pin Properties](#) on page 87
- [Setting Up Symbol Pin Properties](#) on page 97
- [Adding Package Pin Properties](#) on page 128
- [Adding Symbol Pin Properties](#) on page 136
- [Adding Symbol Pin Properties for Individual Bits of a Vector Pin](#) on page 137.

## **Modifying Pin-to-Pin Spacing on Symbols**

### ***Available from:***

Modify Values

The *Modify Values* option enables you to modify pin-to-pin spacing on a symbol. For more information, see [Modify Column Values](#) on page 447

## **Turning Validation Checks On or Off**

### ***Available from:***

Setup

You can control the on-load and on-save validation through the Setup. For more information, see [Validation](#) on page 86

---

## List of Valid Values in Part Developer

---

### Supported Characters in Cell Names

- a-z
- 0-9
- -
- -
- /



The front-to-back flow does not support library, cell, or view names in uppercase or mixed case.

### Unsupported Characters for Cell Names

- \
- :
- "
- '
- <
- Space

**Note:** To use an unsupported character in part names specified using the *Save As* option, you need to add the character to the `SaveAs -ValidCharSet` list in the `profile.prop` file located at `<install_dir>/share/cdssetup/LMAN`. However, while adding space to the `ValidCharSet` list, ensure that you add space in the middle of the list. A leading or trailing space is ignored.

## Unsupported Characters for Property Values in PTF File

|

"

'

## Unsupported Characters for PART\_NAME Property Values

- ~
- `
- !
- \*
- (
- )
- -
- +
- =
- |
- \
- }
- ]
- {
- [
- :
- ;
- "
- '

- <
- >
- .
- ?

## **Supported Characters for PACK\_TYPE Property Values**

- A-Z
- 0-9
- (
- )
- -
- \_

## **Supported Characters for Property Names in Packages/ Symbols/Package Pins/Symbol Pins**

- A-Z
- \_
- 0-9
- \$
- -

## **Unsupported Characters for Packages/Symbols/Package Pins/Symbol Pin Property Values**

- '
- "
- !

## Supported Characters for Differential Pair Names

- A-Z
- 0-9
- #
- \$
- -
- &
- @
- %
- ^
- /
- "
- ;

## Pin Naming

Pins should be designated with functional names. Each pin name must be unique to that symbol and must have a matching entry in the `chips.prt` file. Typically, a pin name must be alphanumeric, but you can have numbers as pin names for scalar pins. The other characters that are supported by Design Entry HDL as valid characters in pin names are as follows:

- -
- #
- \$
- %
- +
- =
- |
- ?

## Part Developer User Guide

### List of Valid Values in Part Developer

---

- ^
- -
- .
- (
- )
- /
- {
- }
- [
- ]



#### *Caution*

***For pins that have ( or ) in their names, hlibftb reports errors. However, you can use such pins in Design Entry HDL schematics by turning off the multi\_format\_vector option off.***

The following are not valid for pin names:

- All extended character sets
- ;
- !
- <
- >
- :
- \
- ‘
- “
- ,
- \*
- ~

**Note:** Low-asserted pin names should be denoted only with an asterisk (\*) (for example, OE\*) or \_N (for example, OE\_N). All low-asserted pins should appear as bubbles and not straight pin stubs.



You can use the PinName\_Invalid\_CharacterSet CPM directive to specify additional characters to be treated as invalid. For example, if you want to disallow the use of \_, (, and ) characters in pin names, specify PinName\_Invalid\_CharacterSet '\_() ' in your CPM file.

## Pin Name/Number Length

The maximum number of characters in a pin name or number is 30. For more details, see *Allegro/APD User Guide: Getting Started* in Cadence documentation system.

## Primitive Name Length

The maximum number of characters in a primitive name is 30. For more details, see *Allegro/APD User Guide: Getting Started* in Cadence documentation system.

## Body Section Property Name Length

The maximum length of a body section property name is 2048.

## Body Section Property Value Length

The maximum length of a body section property value is 8192.

## PORT\_ORDER Value Length

The maximum length of a PORT\_ORDER property value is 8192.

## Valid Range Notations

The following are examples of valid range specifications in Design Entry HDL schematics:

## Part Developer User Guide

### List of Valid Values in Part Developer

---

<b>Notation Examples</b>	<b>Description</b>
<10 downto 0>	11-bit descending range
<10..0>	11-bit descending range
<0 to 10>	11-bit ascending range
<10:0>	Colon (:) is the same as <code>downto</code>
<0:10>	Colon (:) works like <0 to 10>
<size-1:0>	Parameterized descending range
<0 to size-1>	Parameterized ascending range

The following examples are invalid in Design Entry HDL schematics:

<b>Notation Examples</b>	<b>Description</b>
<10 to 0>	Illegal. <code>to</code> must be an ascending range.
<0 downto 10>	Illegal. <code>downto</code> must be a descending range

## NMP

Related to name spacing in VHDL and Verilog. For more information, see *Cadence Application Infrastructure User Guide* in Cadence documentation system.

## **Part Developer User Guide**

### List of Valid Values in Part Developer

---

---

## Pin Types

---

Part Developer supports 14 pin types. The following table lists these pin types, their basic descriptions, and the minimum chips.prt definition that Part Developer uses to assign a pin type when loading a part.

Pin Type	Description	Minimum chips.prt Definition
ANALOG	The ANALOG pin type is assigned to certain pins of analog components.	PIN_TYPE='ANALOG';
BIDIR	A BIDIR pin is an input/output pin.	BIDIRECTIONAL='TRUE';
INPUT	An input pin is a pin to which you apply a signal. For example, pins 1 and 2 on the 74LS00 NAND gate are input pins.	INPUT_LOAD='(-0.01,0.01)';
OUTPUT	An output pin is a pin to which the part applies a signal. For example, pin 3 on the 74LS00 NAND gate is an output pin.	OUTPUT_LOAD='(1.0,-1.0)';
TS	A tristate pin has three possible states: low, high, and high impedance. When it is in its high-impedance state, a tristate pin looks like an open circuit. For example, the 74LS373 latch has tristate pins.	OUTPUT_TYPE='(TS,TS)';

## Part Developer User Guide

### Pin Types

---

TS_BIDIR	A tristate bidirectional pin. A bidirectional pin is either an input pin or an output pin. For example, pin 2 on the 74LS245 bus transceiver is a bidirectional pin. The value at pin 1 (an input pin) determines the active type of pin 2, as well as others.	OUTPUT_TYPE='(TS,TS)'; BIDIRECTIONAL='TRUE';
OC	An open collector gate omits the collector pull-up. Use an open collector to make "wired-OR" connections between the collectors of several gates and to connect with a single pull-up resistor. For example, pin 1 on the 74LS01 NAND gate is an open collector gate.	OUTPUT_TYPE='(OC,AND)';
OC_BIDIR	An open collector bidirectional pin.	OUTPUT_TYPE='(OC,AND)'; BIDIRECTIONAL='TRUE';
OE	An open emitter gate omits the emitter pull-down. The appropriate resistance is added externally. ECL logic uses an open emitter gate and is analogous to an open collector gate. For example, MC10100 has an open emitter gate.	OUTPUT_TYPE='(OE,OR)';
OE_BIDIR	An open emitter bidirectional pin.	OUTPUT_TYPE='(OE,OR)'; BIDIRECTIONAL='TRUE';
POWER	A power pin expects either a supply voltage or ground. For example, on the 74LS00 NAND gate, pin 14 is VCC and pin 7 is GND.	PINUSE='POWER';
NC	A no-connect, pin-on-body pin.	PINUSE='NC';
UNSPEC	A pin with no specific function. This pin type is often used for connectors and passive devices.	PINUSE='UNSPEC';

## Part Developer User Guide

### Pin Types

---

GROUND	A ground pin.	If the pin is in the body section  GROUND_NETS = ' <i>pin name</i> ';  POWER_PINS = (< <i>pin name</i> >:<pin number>);  If the pin is in the pin section '< <i>pin name</i> >':  PIN_NUMBER = ('< <i>pin number</i> >');  PINUSE = 'GROUND';
--------	---------------	--

**Note:** When a component is taken through the front-to-back flow, Allegro PCB Editor translates all the pin type definitions to PINUSE definitions according to the following mapping:

---

Part Developer Pin Type	PINUSE Value
ANALOG	UNSPEC
BIDIR	BI
INPUT	IN
OUTPUT	OUT
TS	TRI
TS_BIDIR	BI
OC	OCA
OC_BIDIR	BI
OE	OCL
OE_BIDIR	BI
POWER	POWER
NC	NC
UNSPEC	UNSPEC
GROUND	GROUND

---

## **Part Developer User Guide**

### **Pin Types**

---

# Index

---

## Symbols

[ ] in syntax [20](#)  
\$LOCATION [85, 108](#)  
\$SPLIT\_INST\_NAME [108](#)

## A

add physical pins manually [56](#)  
adding global pins [120](#)  
adding logical pins [112](#)  
    Add Pin dialog  
        benefits [112](#)  
    directly through Package Editor [117](#)  
    directly through Symbol Editor [119](#)  
associate footprints [47](#)  
asymmetrical part [108](#)  
Auto Expand Bus [94](#)

## B

BaseLine on Save [333](#)  
body section property name length [566](#)  
body section property value length [566](#)  
brackets in syntax [20](#)

## C

CDS\_SITE [341](#)  
    setting up [341](#)  
Cell Editor tree pane [41](#)  
Cell Editor user interface [40](#)  
    Cell Editor tree [41](#)  
Check Assert [92](#)  
Check Dir [92](#)  
Check IO [91](#)  
Check Load [90](#)  
Check Output [92](#)  
CLASS [46](#)  
configuring  
    Class values [348](#)  
    data translation [351](#)  
    default properties [349](#)

pin text position defaults [360](#)  
pin type translation [343](#)  
placeholder property value [342](#)  
predefined headers for CSV import [357](#)  
predefined headers for text import [359](#)  
RefDes Prefix [348](#)  
sheet sizes [347](#)  
symbol pin property alignment [362](#)  
symbol pin property rotation [362](#)  
symbol property alignment [361](#)  
to import pins with square brackets in  
    basenames when  
        VectorSqrBracket is set to 1 [361](#)  
to launch external tools [344](#)  
to use square brackets in vector pin  
    names [360](#)  
conventions  
    for user-defined arguments [20](#)  
creating new cells [110](#)  
creating packages [121](#)  
    from symbols [122](#)  
    using Package Editor [122](#)  
creating parts  
    from PDFs [145](#)  
        copying data directly into pin  
            grid [147](#)  
        copying from a spreadsheet [149](#)  
creating symbols [134](#)  
    from a package [135](#)  
    through Symbol Editor [135](#)  
Creation - Date/Time [333](#)  
Creation - Path [334](#)  
Creation - User [334](#)  
custom shapes  
    aligning [185](#)

## D

Default Property Height [95](#)  
dialog box  
    Add Logical Part [440](#)  
    Add Physical Part [440](#)  
    Add Physical Pin Numbers [418](#)  
    Add Pin [410](#)  
        accessing through Package

Editor [112](#)  
 accessing through Symbol Editor [115](#)  
 Add Properties [417](#)  
 Annotate Generics [438](#)  
 Annotate Parameters [439](#)  
 Browse Alt Symbol [436](#)  
 Browse Jedec Type [436](#)  
 Delete Package Pin [416](#)  
 Delete Package Pin Property [418](#)  
 Delete Symbol Pin [419](#)  
 Delete Symbol Pin Property [422](#)  
 Distribute Pins [409](#)  
 Edit Functions [408](#)  
 Edit Global Mapping [446](#)  
 Extracted Shape Name [416](#)  
 Filter Rows [446](#)  
 Find & Replace [448](#)  
 Generate Symbol [443](#)  
 Import and Export Wizard [426](#)  
 Lib  
     Cell  
         View - Select Model [425](#)  
     Map Pin Name and Text [421](#)  
     Modify Column Values [447](#)  
     Modify Model [425](#)  
     Modify Package [424](#)  
     Modify Pin [444](#)  
     Modify Properties [442](#)  
     Move Pin [423](#)  
     New Cell [407](#)  
     New Custom Shape [451](#)  
     New Pin Shape [451](#)  
     Open Cell [408](#)  
     Open Custom Shape [451](#)  
     Open Pin Shape [451](#)  
     Open Project [407](#)  
     Rename [415](#)  
     Rename Package Pin Property [417](#)  
     Rename Physical Part [441](#)  
     Rename Pin [416](#)  
     Rename Symbol Pin Property [422](#)  
     Save All [437](#)  
     Save As [442](#)  
     Select Footprint [436](#)  
     Select Model [425](#)  
     Select Package [423](#)  
     Select Package to Associate [447](#)  
     Select Value to be Annotated [440](#)  
     Specify The Symbol Size [448](#)  
     Symbol Pin Attributes [419](#)  
     Symbol Pin Property Attributes [441](#)  
     Template Application Wizard [452](#)  
     Text Properties [452](#)  
     die file import  
         conversion details [278](#)  
         ECO [294](#)  
         steps [278](#)  
     differential pairs  
         autocreating through the Package Editor [129](#)  
         creating at a library level [129](#)  
         creating from selected pins [132](#)  
         points about Part Developer support [128](#)  
         properties [128](#)  
         removing [132](#)  
         specifying a naming convention [130](#)  
     directives  
         CPM  
             Default\_Diffpair\_Value [131](#)  
             DiffPair\_Recognition\_Rules [130](#)  
             Import\_AllegroFtpprint\_DefaultPinType [277, 344](#)  
             Import\_APD\_strippinnum [233](#)  
             Import\_Csv\_ApplyVectorConversion [361](#)  
             Import\_Csv\_LowassertFlag [249](#)  
             Import\_Csv\_Replace\_assertion  
                 'assertion' [358](#)  
             Import\_Csv\_Replace\_assertionchar  
                 'assertion\_char' [358](#)  
             Import\_Csv\_Replace\_jedectype  
                 'jedec\_type' [358](#)  
             Import\_Csv\_Replace\_loadsetupfile  
                 'load\_setupfile' [358](#)  
             Import\_Csv\_Replace\_packagename  
                 'package\_name' [358](#)  
             Import\_Csv\_Replace\_pinlocation  
                 'pin\_location' [358](#)  
             Import\_Csv\_Replace\_pinname [358](#)  
             Import\_Csv\_Replace\_pinnumber [358](#)  
             Import\_Csv\_Replace\_pinposition  
                 'pin\_position' [358](#)  
             Import\_Csv\_Replace\_pintype [358](#)  
             Import\_Csv\_Replace\_symbol  
                 'symbol' [358](#)  
             Import\_DML\_Braces\_TreatedAs\_Vector [279](#)  
             Import\_DML\_Pins\_DefaultVector [279](#)

Import_FPGA_Braces_TreatedAs_Vector	<u>260</u>	363
Import_IBIS_With_Dmlcheck	<u>281</u>	Symbol_Property_Stacking_Offset <u>364</u>
Import_IBIS_With_Ibischk4	<u>281</u>	Symbol_ValidateSoftPropValue <u>34</u>
Import_IBIS_With_Unchanged_ModuleName	<u>281</u>	<u>2</u>
Import_Text_Braces_TreatedAs_Vector	<u>266</u>	VectorSqrBracket <u>360</u>
Import_Text_DefaultPinType	<u>344</u>	CSV
Import_Text_Pins_DefaultVector	<u>2</u> <u>66</u>	Import_Csv_Replace_DIFFPAIRPIN SNEG 'DIFF_PAIR_PINS_NEG' <u>35</u> <u>8</u>
MULTI_FORMAT	<u>360</u>	Import_Csv_Replace_DIFFPAIRPIN SPOS 'DIFF_PAIR_PINS_POS' <u>35</u> <u>8</u>
PinName_Invalid_CharacterSet	<u>56</u> <u>6</u>	Import_Csv_Replace_pinshape 'pin_shape' <u>358</u>
PinName_MaxLength	<u>478</u>	profile files
Symbol_MaxSymSize	<u>93</u>	EndRowNumber <u>270</u>
Symbol_PinProperty_Alignment_Bottom	<u>362</u>	GenerateSymbol <u>270</u>
Symbol_PinProperty_Alignment_Left	<u>362</u>	OtherDelimiter <u>270</u>
Symbol_PinProperty_Alignment_Right	<u>362</u>	PinNumasPinname <u>270</u>
Symbol_PinProperty_Alignment_Top	<u>362</u>	SetFirstRowAsHeader <u>270</u>
Symbol_PinProperty_Rotation_Bottom	<u>362</u>	StartRowNumber <u>270</u>
Symbol_PinProperty_Rotation_Left	<u>362</u>	TextDelimiters <u>270</u>
Symbol_PinProperty_Rotation_Right	<u>362</u>	TextQualifier <u>270</u>
Symbol_PinProperty_Rotation_Top	<u>362</u>	TreatConsecutiveAsOne <u>270</u>
Symbol_PinPropertySetup_Alignment_Apply	<u>362</u>	translate.cpm
Symbol_PinPropertySetup_Rotation_Apply	<u>362</u>	CONCEPT_SYMBOL_SHEET_A <u>3</u> <u>47</u>
Symbol_Pintext_LeftRight_XOffset	<u>360</u>	CONCEPT_SYMBOL_SHEET_B <u>3</u> <u>47</u>
Symbol_Pintext_LeftRight_YOffset	<u>360</u>	CONCEPT_SYMBOL_SHEET_C <u>3</u> <u>47</u>
Symbol_Pintext_TopBottom_XOffset	<u>360</u>	CONCEPT_SYMBOL_SHEET_D <u>3</u> <u>47</u>
Symbol_Pintext_TopBottom_YOffset	<u>360</u>	CONCEPT_SYMBOL_SHEET_E <u>3</u> <u>47</u>
Symbol_Property_Outline_OffsetBottom	<u>363</u>	CONCEPT_SYMBOL_SHEET_F <u>3</u> <u>47</u>
Symbol_Property_Outline_OffsetLeft	<u>363</u>	MENTOR_VERSION <u>284</u>
Symbol_Property_Outline_OffsetRight	<u>363</u>	Distribute Pins <u>126</u>
Symbol_Property_Outline_OffsetTop		

## E

ECO	
Capture	<u>286</u>
CSV	<u>289</u>
EDAXML	<u>287</u>
methodology	<u>225</u>

Si2 PinPak [288](#)  
Synopsis PTM [289](#)  
Error [333](#)  
examples  
    importing a text file [267](#)

## F

filter  
    creating [174](#)  
    removing [175](#)  
    using [124, 174](#)  
Find Filter  
    All Off button [65](#)  
    All On button [65](#)  
    Find By Name [65](#)  
    graphical object search [65](#)  
    grouped object search [65](#)  
    text search [65](#)  
font editing  
    additional shortcut keys [38](#)  
    Decrease Font button [37](#)  
    Increase Font button [37](#)  
footprint  
    importing [277](#)  
    modifying [161](#)  
full-screen editing [32](#)

## G

global pins [54](#)  
    adding [120](#)  
    moving from Logical Pins grid to Global  
        Pins grid [115](#)  
Graphic Editor  
    Align [33](#)  
    Canvas [34](#)  
    Draw [32](#)  
    Font [37](#)  
    Layout [34](#)  
    Nudge [33](#)  
    Rotate [34](#)  
    Zoom/Pan [31](#)  
grid density [93](#)  
grid size [93](#)  
grid usability tips [558](#)  
    adding PIN\_GROUP and other pin-level  
        properties [560](#)  
    creating pass-through and bubble

    pins [559](#)  
    creating vector pins from scalar  
        pins [559](#)  
    filtering rows [559](#)  
    modifying column values [558](#)  
    modifying pin-to-pin spacing on  
        symbols [560](#)  
    setting the symbol origin [559](#)  
    turning validation checks on/off [560](#)  
grids  
    Shape Editor [180](#)  
    Symbol Editor [66](#)  
GROUND pins [54](#)

## I

import and export  
    Capture export [299](#)  
    Capture import [234](#)  
    CSV [246](#)  
    die text import [278](#)  
    ECO process [225](#)  
    EDAXML export [304](#)  
    EDAXML import [240](#)  
    footprint [277](#)  
        ECO [293](#)  
        procedure [277](#)  
    methodology [224](#)  
    Si2 PinPak import [243](#)  
    Synopsis PTM import [255](#)  
    text file import [264](#)  
        example [267](#)  
        procedure [271](#)

    Verilog import [257](#)  
    VHDL import [258](#)  
instantiation and packaging  
    con2con checks [314](#)  
    hlibftb checks [315](#)  
Interface Comparator  
    running [321](#)  
italics in syntax [20](#)

## J

JEDEC\_TYPE [47](#)

### L

LastModification - Date/Time [334](#)  
LastModification - Path [334](#)  
LastModification - User [334](#)  
library management use model  
    designer [24](#)  
load checks [90](#)  
logical and physical parts  
    modifying [160](#)  
logical parts [45](#)  
logical pins  
    adding [112](#)  
    modifying [155](#)  
Low Assert Shape [97](#)  
low assertion  
    - pin name prefix in chips [84](#)  
    Additional Read [83](#)  
    pin text [83](#)  
    Read/Write [83](#)  
    setup [83](#)  
low-asserted  
    - pin name prefix in chips [84](#)  
    Additional Read [83](#)  
    pin text [83](#)  
    Read/Write [83](#)  
    setup [83](#)

### M

Minimum Pin Spacing [97](#)  
Minimum Size - Height [94](#)  
Minimum Size - Width [94](#)  
Modification Tips [173](#)  
modification tips  
    modifying multiple rows and  
        columns [173](#)  
    directly in the grid [174](#)  
    using the shortcut menu [173](#)

### N

Name [333](#)  
NC pins [54](#)  
    adding [120](#)  
NMP [567](#)

### O

Output Load [90](#)

### P

Package Editor [44](#)  
    General [45](#)  
    Global Pins [54](#)  
    Logical Pins [48](#)  
    Physical Pins [55](#)  
package pin properties  
    deleting from all pins [167](#)  
    deleting from specific pins [167](#)  
    renaming [168](#)  
package pins  
    adding [48](#)  
    modifying [158](#)  
    renaming [156](#)  
package properties [47](#)  
    entering [123](#)  
    modifying [160, 161](#)  
packages  
    creating [121](#)  
    modifying [159](#)  
part creation methodology [109](#)  
Part Developer menus [27](#)  
    Edit [28](#)  
    File [27](#)  
    Graphic Editor [31](#)  
    Tools [29](#)  
    View [29](#)  
Part Developer registry entry  
    cleaning [342](#)  
Part Developer work environment [25](#)  
part logging and versioning  
    adding comments to the revision  
        log [336](#)  
    major and minor revision number  
        updates [337](#)  
    restarting [337](#)  
    Revision Editor [332](#)  
    starting [334](#)  
    stopping [337](#)  
part templates  
    verifying a part against a template [103](#)  
part types [107](#)  
    asymmetrical [108](#)  
    split [108](#)

symmetrical [107](#)  
PCB Librarian XL only features [19](#)  
physical parts [45](#)  
pin location [98](#)  
pin name format for bus [97](#)  
pin name height [96](#)  
pin name length [566](#)  
pin name notation difference between UI and chips [84](#)  
pin name notation in chips [84](#)  
pin number length [566](#)  
pin order  
    retrieving [121](#)  
    setting [121](#)  
pin properties  
    modifying [172](#)  
pin swapping  
    *See also* split parts [89](#)  
pin text color [96](#)  
pin text height [96](#)  
PIN\_DELAY [92](#)  
    specifying [126](#)  
PIN\_GROUP  
    specifying through setup [89](#)  
pin\_location  
    import text values [265](#)  
pins  
    deleting [168](#)  
    deleting across all packages and symbols [170](#)  
    modifying across all packages and symbols [159](#)  
    renaming [156](#)  
    renaming across all packages and symbols [156](#)  
    setting and retrieving pin order [121](#)  
PINUSE  
    correlation with pin types [571](#)  
PORT\_ORDER value length [566](#)  
POWER pins [54](#)  
    adding [120](#)  
primitive name length [566](#)  
PSMPATH [47](#)

**R**

RefDes Prefix [46](#)  
Revision Editor [332](#)  
revision log  
    adding comments [336](#)

viewing [335](#)  
Revision-Major [333](#)  
Revision-Minor [333](#)

**S**

scalar pins to vector  
    converting [167](#)  
setting up Part Developer [81](#)  
setup  
    low-asserted pins [83](#)  
    package defaults [86](#)  
    package pin defaults [87](#)  
    PTF defaults [98](#)  
    RefDes Prefix [86](#)  
    shape defaults [181](#)  
    split parts [83, 85](#)  
    symbol pin defaults [96](#)  
    symbol pin properties [97](#)  
    symbol properties [93](#)  
Setup Options tree [82](#)  
Shape Editor  
    grid settings [180](#)  
    object movement with arrow keys [181](#)  
shapes  
    attaching custom shapes [184](#)  
    creating [182](#)  
    extracting [183](#)  
    inserting custom shapes [185](#)  
    replacing [183](#)  
    setting up [181](#)  
    types [177](#)  
Sheet Size [93](#)  
Show Dot as Filled [97](#)  
SI Model Interface Comparison  
    running [329](#)  
Snap to Grid  
    behavior [35](#)  
split parts [108](#)  
    creating [127](#)  
    pin swapping [85](#)  
    setup [85](#)  
SPLIT\_INST [108](#)  
SPLIT\_INST\_NAME [85](#)  
Status - Major [333](#)  
Stub Length [97](#)  
supported characters  
    cell name [561](#)  
    differential pair name [564](#)  
PACK\_TYPE property value [563](#)

- 
- pin name [564](#)
  - property names
    - package [563](#)
    - package pin [563](#)
    - symbol [563](#)
    - symbol pin [563](#)
  - SWAP\_INFO [85](#)
  - Symbol Editor [57](#)
    - General page [58](#)
    - grid settings [66](#)
    - Logical Pins grid [60](#)
    - object movement with arrow keys [67](#)
    - Symbol Pins page [60](#)
  - symbol or package pins
    - deleting [168](#)
  - Symbol Outline [94](#)
  - symbol pin properties [98](#)
    - modifying attributes [173](#)
  - symbol pins
    - modifying [157](#)
    - renaming [156](#)
  - symbol properties [95](#)
    - deleting from all pins [172](#)
    - deleting from specific pins [172](#)
    - modifying [171](#)
    - position of the first property [363](#)
    - renaming [173](#)
    - stacking [215](#)
    - stacking offset [364](#)
  - symbol property templates
    - applying to a symbol [219](#)
    - applying to all symbols [219](#)
    - applying to the setup [220](#)
    - components [213](#)
    - creating [214](#)
    - extracting [220](#)
    - location of extracted properties [220](#)
    - opening [217](#)
    - position of extracted properties [220](#)
  - symbol text
    - modifying [171](#)
  - symbols
    - creating [134](#)
    - creating packages [122](#)
    - creating using Symbol Editor [135](#)
    - direct generation from a package [135](#)
    - modification types [170](#)
    - modifying outlines [171](#)
    - modifying properties [171](#)
    - modifying the symbol text [171](#)
    - moving pins [171](#)
  - symmetrical parts [107](#)
  - System Unit [93](#)
- ## T
- Text Attributes - Color [94](#)
  - Text Attributes - Height [94](#)
  - Text Attributes - Rotation [95](#)
  - text file
    - importing [264](#)
  - trailing asterisks in pin names [84](#)
  - translation
    - import data [351](#)
    - pin types [343](#)
- ## U
- Unknown Loading [92](#)
  - unsupported characters
    - cell name [561](#)
    - PART\_NAME property value [562](#)
  - property names
    - package pin [563](#)
    - packages [563](#)
    - symbols [563](#)
  - Use Pin Name as Pin Text [96](#)
- ## V
- valid range notations [566](#)
  - VCC pins
    - adding [120](#)
  - Vector Bit Mask [96](#)
  - verifying parts
    - advanced view checks [316](#)
    - instantiation and packaging [314](#)
      - con2con checks [314](#)
      - hlibfb checks [315](#)
    - verify with templates [317](#)
      - grid checks [318](#)
      - minimum size checks [319](#)
      - outline checks [319](#)
      - pin checks [318](#)
      - pin load checks [317](#)
      - property checks [317](#)
      - symbol checks [317](#)
    - Verilog compilation [316](#)
    - VHDL compilation [316](#)

## Part Developer User Guide

---

view verification [313](#)  
Verilog wrappers and map files  
    creating a map file [201](#)  
    creating a wrapper file [207](#)  
    deleting [211](#)  
    modifying [211](#)  
    renaming [211](#)  
vertical bars in syntax [20](#)  
VHDL wrappers and map files  
    creating [189](#)  
    creating a wrapper [195](#)  
    deleting [200](#)  
    map file editor [67](#)  
    modifying [200](#)  
    renaming [200](#)  
    wrapper file editor [71](#)